80287 SUPPORT LIBRARY REFERENCE MANUAL

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Order Number: 122129-001

intel

80287 SUPPORT LIBRARY REFERENCE MANUAL

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Order Number: 122129-001

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

A1079/1083/3K/JAY/KH

REV.	REVISION HISTORY	DATE	APPD.
-001	Original issue.	10/83	R.N.

PREFACE

This manual is a programmer's guide to the 80287 Support Library. The Library is a package of software tools that enable the 80287 Numeric Data Processor (NDP) to provide a full range of floating-point capabilities to ASM286, PL/M-286 and Pascal-286 programmers.

This manual is a reference guide, designed for use when writing code for ASM286, PL/M-286 and Pascal-286 programs. The documentation for each procedure and function is as self-contained as practical to avoid excessive page-flipping.

For an initial overview of the library, read Chapters 1 and 2 and the first parts of Chapters 3, 4, and 5. Read Chapter 6 if you are concerned with compliance to the IEEE Standard. Pascal-286 programmers should read Appendix A.

To find specific facts about individual procedures and functions, look in the reference pages, headed by large names, at the ends of Chapters 3, 4, and 5.

Note that the reference pages contain material describing unusual values, such as denormals, unnormals, and pseudo-zeros. Remember these values are very uncommon and never occur for most applications.

Appendixes D and E are quick reference material for experienced users of the 80287 Support Library.

Related Publications

The following manuals are useful when reading this manual. Particularly crucial is the documentation on the 80287 itself, found in the *ASM286 Language Reference Manual*, order number 121924.

- *Introduction to the iAPX 286*, order number 210308
- *PL/M-286 User's Guide*, order number 121945
- *Pascal-286 User's Guide*, order number 122051
- *iAPX 286 Utilities User's Guide*, order number 121934
- *iAPX 286 System Builder User's Guide*, order number 121935
- *ASM286 Macro Assembler Operating Instructions*, order number 121925
- *iAPX 286 Programmer's Reference Manual*, order number 210498
- *iAPX 286 Programmer's Reference Manual Numerics Supplement*, order number 122164

System Requirements

The 80287 Support Library is designed to function in an iAPX 286/20 environment. This environment consists of an iAPX 286 microprocessor and an 80287 Numeric Data Processor.

Notational Conventions

The programming examples in this manual are presented as code that can be inserted intact into a PL/M-286 or ASM286 program. The examples contain no formulas of general syntax.

Throughout this manual the term Numeric Processor Extension (NPX) refers to the 80287 component. The term Numeric Data Processor (NDP) refers to any system that contains an NPX.

The invocation examples follow the conventions commonly used in Intel manuals:

input lines	In interactive examples, user input lines are printed in white on black to differentiate them from system output.
<cr>	Indicates a carriage return.

CONTENTS

FIGURES

CHAPTER 1 INTRODUCTION

The 80287 Support Library greatly facilitates the use of floating-point calculations in ASM286, PL/M-286 and Pascal-286.

The interface libraries (80287.LIB, and NUL287.LIB) allow a choice of run-time environments for modules containing floating-point code.

The decimal conversion library (DC287.LIB) aids the translation between human-readable numbers of many decimal formats and binary numbers in all of the formats supported by the Numeric Processor Extension (NPX). It also provides translation between binary formats.

The common elementary function library (CEL287.LIB) provides a useful assortment of transcendental functions, rounding functions, and other common functions involving floating-point numbers.

The error handler module (EH287.LIB) makes it easy to write interrupt procedures that recover from floating-point error conditions.

Why the Support Library Is Needed

Three areas in which many applications require support software to overcome functional limitations of the NDP are the following:

- First, the NDP is, exclusively, a binary arithmetic machine. Although the NDP manipulates a variety of data types, it cannot operate directly on floating-point decimal quantities. Software must be invoked to convert decimal inputs to binary formats and binary answers to decimal output.
- Second, the 80287 instruction set is limited in its implementation of elementary functions. There is a tangent function, for example, but no sine, and the tangent is defined only for inputs between 0 and $\pi/4$. The implementation of a full set of elementary functions with complete domains is left to software.
- Third, the handling of unmasked exceptions can complicate application software. Although the masked mode of error detection and recovery is adequate for most applications, customized error recovery is sometimes needed. Deciphering the NPX status information and restoring the NPX to a recovery state can be quite difficult. The 80287 Support Library software can make the task easier.

System Software Needed to Use the Library

Because the Support Library is called by ASM286, PL/M-286, and Pascal-286 programs, you will need either the ASM286 assembler or the PL/M-286 or Pascal-286 compiler.

You will need to compile your PL/M-286 and Pascal-286 modules under the MEDIUM or LARGE models of computation, because all of Support Library procedures are FAR procedures.

You will also need the 80286 utility BND286 to convert the object modules generated by your programs into programs that can be loaded or programmed into memory and executed. (If you use EH287.LIB and your own interrupt routine, you will need to use the 80286 system builder, BLD286.)

For consistency in the examples in this manual, BND286 and the Support Library are assumed to be on drive 0, all user-supplied files are assumed to be on drive 1.

No run-time software is needed for the Support Library. Once you have obtained the final executable program, the program will run on any NDP system.

CHAPTER 2
THE INTERFACE LIBRARIES

Overview

The 80287 Support Library contains two interface libraries, 80287.LIB and NUL287.LIB. By selecting one of these libraries to appear in your BND286 statement, you can choose the run-time environment of your program: either with the 80287 component or with no NPX capability.

The interface libraries also contain the INIT87 and INITFP procedures, which initialize the NPX under the various environments, and FILTER(TQ_312), a null Boolean function used when an application program references FILTER, but does not use EH287.LIB.

Initializing the NPX

When power is first applied to an NDP system, the NPX must be brought to a known state. This is accomplished by calling one of the 80287 initialization procedures.

We have provided two 80287 initialization procedures, INIT87 and INITFP, in each of the interface libraries. The 80287.LIB versions initialize the 80287 component and set the 80287 control word. The NUL287.LIB versions satisfy 80287 externals with null procedures and should be used only with software that contains no NPX instructions.

The difference between INIT87 and INITFP is the 80287 control word default settings. INIT87 masks all exceptions. INITFP masks all exceptions except I. Although you may later modify the control word, you must first call either INIT87 or INITFP.

PL/M-286 Use of INIT87 or INITFP

To use INIT87 in a PL/M-286 program, you must invoke the procedure before any floating-point arithmetic is performed. This can be done with the built-in procedure

```
CALL INIT$REAL$MATH$UNIT;
```

Following is the declaration of INITFP,which must appear at the top of your initialization module, and the call, which must be invoked before any floating-point arithmetic is performed.

Because INIT87 and INITFP are FAR procedures, you must compile the PL/M-286 module under the MEDIUM or LARGE controls.

```
INITFP: PROCEDURE EXTERNAL;
END;

CALL INITFP;
```

ASM286 Use of INIT87 or INITFP

Following is the declaration of INIT87, which must appear at the top of your initialization module, and the call instruction, which must appear within the initialization procedure. The statements for INITFP are the same, with INITFP instead of INIT87.

```
; The following line must appear outside all
; SEGMENT-END pairs
   EXTRN INIT87: FAR

   CALL INIT87      ; Set up the NPX
```

Binding Interface Libraries to User Programs

When using the iAPX 286 with the 80287 NDP, you must *bind* the 80287.LIB library to your object modules. If you issue a call to INIT87 or INITFP but have no other floating-point instructions, you should bind NUL287.LIB to your object modules. The interface libraries must appear in the sequence of input modules after all modules that contain 80287 code.

Following is the suggested order for the object modules in your BND286 command:

Your object modules
DC287.LIB if you are using it
CEL287.LIB if you are using it
EH287.LIB if you are using it
80287.LIB or
NUL287.LIB if you are using nonfloating-point application programs.

Examples

Suppose your program consists of two modules, MYMOD1.OBJ and MYMOD2.OBJ, created with either ASM286 or PL/M-286. Issue the following command:

```
-BND286 :F1:MYMOD1.OBJ, &<cr>
        :F1:MYMOD2.OBJ, &<cr>
        :F0:EH287.LIB, &<cr>
        :F0:80287.LIB &<cr>
        OBJECT (:F1:MYPROG.LNK)<cr>
```

If your program calls INIT87 or INITFP but contains no other floating-point instructions, issue the following command:

```
-BND286 :F1:MYMOD1.OBJ, &<cr>
        :F1:MYMOD2.OBJ, &<cr>
        :F0:NUL287.LIB &<cr>
        OBJECT (:F1:MYPROG.LNK)<cr>
```

THE DECIMAL CONVERSION LIBRARY

This chapter describes the use of DC287.LIB, a library of procedures that convert floating-point numbers from one format of storage to another.

Overview of DC287 Procedures

Intel supports three different binary formats for the internal representation of floating-point numbers (see Appendix C). DC287.LIB provides translation between these formats and an ASCII-encoded string of decimal digits.

All DC287 procedure names begin with the three letters *mqc*. This prefix has been added to reduce the chance of conflict between a DC287 name and another name in your program. To make the names more readable, *mqc* prefix appears in lowercase letters throughout this chapter.

The binary-to-decimal procedure mqcBIN_DECLOW accepts a binary number in any of the three formats. Because floating-point numbers have so many output formats, mqcBIN_DECLOW does not attempt to provide a finished, formatted text string. Instead, it provides the "building blocks" for you to use to construct the output string that meets your exact format specification.

The decimal-to-binary procedure mqcDEC_BIN accepts a text string that consists of a decimal number with optional sign, decimal point, and/or power-of-ten exponent. It translates the string into the caller's choice of binary formats.

An alternate decimal-to-binary procedure mqcDECLOW_BIN provides an input format similar to the output of mqcBIN_DECLOW. The low-level input interface is provided for callers who have already broken the decimal number into its constituent parts.

The mqcLONG_TEMP, mqcSHORT_TEMP, mqcTEMP_LONG, and mqcTEMP_SHORT procedures convert floating-point numbers between the longest binary format, TEMP_REAL, and the shorter formats. The conversions are performed so that the outputs are NaNs if and only if the inputs are NaNs. This improves upon the conversion algorithm used by the 80287 when it loads shorter formats to and from its stack.

DC287 contains alternate public names for its procedures. The names, listed in Appendix F, are used by some Intel translators.

Declaring DC287 Procedures in ASM286 Programs

In each source program module that calls a DC287 procedure, you must declare that procedure as external before it is used.

Following are the declarations you should make at the top of your ASM286 program. You should include only those procedures you use.

All DC287 procedures must be declared FAR.

This ASM286 code also includes declarations that ease the construction of parameters. The ASM286 examples given with the procedures later in this chapter assume that these declarations have been made.

For easier referencing, the procedure declarations in the examples in the individual functions have been duplicated. You should not duplicate the declarations in your programs.

```
; ASM286 declarations for using the DC287 decimal
; conversion library

; the following EXTRN statements must appear outside of
; all SEGMENT-ENDS pairs.

EXTRN mqcBIN_DECLOW: FAR
EXTRN mqcDEC_BIN: FAR
EXTRN mqcDECLOW_BIN: FAR
EXTRN mqcLONG_TEMP: FAR
EXTRN mqcSHORT_TEMP: FAR
EXTRN mqcTEMP_LONG: FAR
EXTRN mqcTEMP_SHORT: FAR

SHORT_REAL EQU 0
LONG_REAL  EQU 2
TEMP_REAL  EQU 3

BIN_DECLOW_BLOCK STRUC
  BIN_PTR       DD ?
  BIN_TYPE      DB ?
  DEC_LENGTH    DB ?
  DEC_PTR       DD ?
  DEC_EXPONENT  DW ?
  DEC_SIGN      DB ?
BIN_DECLOW_BLOCK ENDS

DEC_BIN_BLOCK STRUC
  DD ?    ;  The names of these fields are the same as
  DB ?    ;    the first four fields in BIN_DECLOW_BLOCK.
  DB ?
  DD ?
DEC_BIN_BLOCK ENDS

DECLOW_BIN_BLOCK STRUC
  DD ?     ;  The names of these fields are the same as in
  DB ?     ;    BIN_DECLOW_BLOCK.
  DB ?
  DD ?
  DW ?
  DB ?
DECLOW_BIN_BLOCK ENDS
```

Declaring DC287 Procedures in PL/M-286 Programs

Following are the declarations you should make at the top of your PL/M-286 program. You should include only those procedures you use.

This PL/M-286 code also includes declarations that ease the construction of parameters. The PL/M-286 examples given with the procedures later in this chapter assume that these declarations have been made.

For easier referencing, the procedure declarations in the examples for the individual functions have been duplicated. You should not duplicate the declarations in your programs.

To use DC287.LIB you must compile your PL/M-286 modules under the MEDIUM or LARGE computation models, because DC287.LIB's procedures are FAR procedures.

See the section "Alternate Method of DC287 Error Detection" later in this chapter for a discussion of when you might want to declare the procedures as BYTE procedures (to receive an error status).

```
/* PL/M-286 declarations for using the DC287 decimal
   conversion library */

mqcBIN_DECLOW: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcBIN_DECLOW;

mqcDEC_BIN: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcDEC_BIN;

mqcDECLOW_BIN: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcDECLOW_BIN;

mqcLONG_TEMP: PROCEDURE (LONG_REAL_PTR,TEMP_REAL_PTR)
  EXTERNAL;
  DECLARE (LONG_REAL_PTR,TEMP_REAL_PTR) POINTER;
END mqcLONG_TEMP;

mqcSHORT_TEMP: PROCEDURE (SHORT_REAL_PTR,TEMP_REAL_PTR)
  EXTERNAL;
  DECLARE (SHORT_REAL_PTR,TEMP_REAL_PTR) POINTER;
END mqcSHORT_TEMP;

mqcTEMP_LONG: PROCEDURE (TEMP_REAL_PTR,LONG_REAL_PTR)
  EXTERNAL;
  DECLARE (TEMP_REAL_PTR,LONG_REAL_PTR) POINTER;
END mqcTEMP_LONG;

mqcTEMP_SHORT: PROCEDURE (TEMP_REAL_PTR,SHORT_REAL_PTR)
  EXTERNAL;
  DECLARE (TEMP_REAL_PTR,SHORT_REAL_PTR) POINTER;
END mqcTEMP_SHORT;

DECLARE SHORT_REAL LITERALLY '0';
DECLARE LONG_REAL LITERALLY '2';
DECLARE TEMP_REAL LITERALLY '3';

DECLARE BIN_DECLOW_BLOCK STRUCTURE(
 BIN_PTR POINTER,
 BIN_TYPE BYTE,
 DEC_LENGTH BYTE,
 DEC_PTR POINTER,
```

```
 DEC_EXPONENT INTEGER,
 DEC_SIGN BYTE);

DECLARE DEC_BIN_BLOCK STRUCTURE (
 BIN_PTR POINTER,
 BIN_TYPE BYTE,
 DEC_LENGTH BYTE,
 DEC_PTR POINTER);
```

Stack Use of DC287 Procedures

DC287 requires 176 bytes of the 80286 stack for its internal storage. This amount is allocated in the public STACK segment by the DC287 modules.

All DC287 procedures save the entire state of the 80287 upon entry and restore it upon exit.

Decimal Conversion Library Accuracy

DC287 guarantees that an 18-digit decimal number will remain the same when converted into TEMP_REAL and then back to decimal. However, DC287 cannot guarantee that a TEMP_REAL number will remain the same when converted to decimal and then back to TEMP_REAL. Such accuracy would require arithmetic beyond the TEMP_REAL precision, which is too slow. The loss of accuracy is at worst less than 3 bits.

The IEEE Standard (described in Chapter 6) specifies the accuracy for decimal conversions of SHORT_REAL and LONG_REAL numbers. Because DC287 uses TEMP_REAL arithmetic to perform its conversions, it exceeds the IEEE specifications.

Following is the range of SHORT_REAL and LONG_REAL numbers for which the IEEE standard specifies best possible conversion accuracy for conversion both to and from the decimal format. The maximum number of decimal digits for best accuracy is given in the second column; the maximum magnitude for the decimal exponent is given in the third column. The exponents are given assuming the decimal point is to the right of the significand. For example, the largest number for totally accurate SHORT_REAL to decimal conversion is 999999999e13. The smallest positive number is 1e-13.

Conversion	Significand	Exponent
Decimal to SHORT_REAL	9 digits	13
Decimal to LONG_REAL	19 digits	27
SHORT_REAL to Decimal	9 digits	13
LONG_REAL to Decimal	17 digits	27

Following is a wider range of numbers for which the best possible conversion is not required. For numbers in the wider range, the IEEE standard allows an error of up to 0.47 times the unit in the last place.

Conversion	Significand	Exponent
Decimal to SHORT_REAL	9 digits	99
Decimal to LONG_REAL	19 digits	999
SHORT_REAL to Decimal	9 digits	54
LONG_REAL to Decimal	17 digits	341

Note that the chart on 3-4 shows the number of decimal digits needed for decimal to binary and binary to decimal conversion routines where representation to the last bit is required. This is not the same as the number of significant digits that can be preserved on a decimal to binary to decimal operation. The number of significant digits is as follows:

SHORT_REAL	6 digits
LONG_REAL	15 digits
TEMP_REAL	18 digits

These are the maximum number of digits that may be requested to ensure correct rounding.

Error Reporting in DC287.LIB

DC287 incorporates the 80287 error-reporting mechanism. Whenever an error is detected, an appropriate 80287 exception bit is set. Using 80287 instructions, you can set your choice of unmasked bits and provide procedures that will be called whenever an unmasked exception is detected. This way, you do not have to test for errors every time you call a DC287 procedure.

Whenever an unmasked exception is detected by any DC287 procedure, it places certain quantities into 80287 storage before calling your specified trap handler.

- The 80287 instruction pointer is set to point at an FLDCW instruction (OPCODE is equal to 016E hex). This FLDCW instruction does not cause any exceptions; therefore, the FLDCW instruction itself does not update the 80287 exception pointer. You can interpret the reference to an FLDCW instruction to mean that DC287 discovered the exception. DECODE procedures in EH287.LIB detect DC287 exceptions in this manner.
- The 2-word selector-offset pair that addresses the input number is placed into the memory operand pointer of the 80287. For mqcDEC_BIN and mqcDECLOW_BIN, this is a pointer into a DC287 internal buffer on the 80287 stack, containing a string of decimal digits stripped for input to DECLOW_BIN. For all other procedures, this is the pointer to the original input binary number.

Alternate Method of DC287 Error Detection

If you prefer to perform error testing yourself, an alternate method is possible. All DC287 procedures return the 80287 exception byte in the AL register.

You can access the AL register in PL/M-286 programs by declaring the DC287 procedures to be of type BYTE. For example, you could change the declaration of mqcBIN_DECLOW to

```
mqcBIN_DECLOW: PROCEDURE (BLOCK_PTR) BYTE EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcBIN_DECLOW;
```

The call to mqcBIN_DECLOW would then take the form

```
STATUS = mqcBIN_DECLOW(@BIN_DECLOW_BLOCK);
```

After executing the above statement, you can test exception conditions by examining STATUS. STATUS must be declared as type BYTE.

Format of Input Decimal Numbers to mqcDEC_BIN

The decimal number that is input to DEC_BIN takes the form of a string of consecutive ASCII bytes of memory, whose pointer is passed in the parameter block. The string may contain blanks after, but not before or within, the number.

The number may begin with a plus sign or a minus sign. Lack of a sign is equivalent to a plus sign.

The optional sign is followed by the significand. The significand is a sequence of one or more decimal digits, with an optional decimal point. The decimal point may appear before, within, or after the decimal digits. The significand must be present and contain at least one decimal digit. The upper limit to the number of digits is determined by the limit of 255 bytes to the entire string; however, only the 20 most significant nonzero digits of a long significand affect the output answer.

The significand is followed by an optional exponent. This consists of one of the letters E, D, or T, followed by an optional plus or minus sign, followed by a string of one or more decimal digits. The signed number indicates the power of ten, which must be multiplied by the significand to yield the desired input number. E, D, and T all have the same meaning (to identify the exponent). They can be in uppercase or lowercase.

Examples of Input Decimal Strings

The following strings all represent the number 100:

```
100
+100
100.
100.00
100.000000000000000000006099216948
1E2
1T2
1d2
1.E2
.1E3
100000d-3
```

Some valid strings for other numbers are

```
3.14159265
-17
-34.25T-009
5.6E7
.001
-.4
-0
```

The following strings are invalid:

```
1 E2     ; embedded space not allowed
E2       ; missing significand
.E2      ; significand must contain at least one decimal digit
```

BIN_DECLOW — mqcBIN_DECLOW (Convert binary number to decimal string, low level interface)

Parameter

BIN_DECLOW_BLOCK_PTR, a double-word that points to a 13-byte block of memory. The block consists of six fields that contain the following variables:

- BIN_PTR, a double-word that points to the input binary number. The amount of memory space occupied by the number is determined by BIN_TYPE.
- BIN_TYPE, a one-byte input code that indicates the type (SHORT_REAL, LONG_REAL, or TEMP_REAL) of the number pointed to by BIN_PTR. The codes for the types are 0, 2, and 3, respectively. The caller must ensure that BIN_TYPE is set to the correct value; any other value gives an undefined result.
- DEC_LENGTH, a one-byte input that indicates the number of decimal digits to be output at DEC_PTR. All one-byte values are legal. If zero is given, only the DEC_EXPONENT and SIGN are returned, with no string at DEC_PTR. A zero DEC_LENGTH combined with unusual inputs yields an Invalid Result error, as described below.
- DEC_PTR, a double-word that points to the memory location that will hold the output string of ASCII decimal digits. Other possible output values are shown in the table beneath the DEC_SIGN heading.
- DEC_EXPONENT, a word variable output by mqcBIN_DECLOW. The decimal point is assumed to be to the right of the digits output at DEC_PTR. DEC_EXPONENT is then the power of ten, which, when multiplied by the number at DEC_PTR, will yield the correct number. For NaNs and infinities, DEC_EXPONENT is 32767. For 0 and -0, DEC_EXPONENT is 0.
- DEC_SIGN, a byte variable output by mqcBIN_DECLOW. It is set to ASCII "+" (2B hex) if the input number is positive, and ASCII "−" (2D hex) if the number is negative. Other possible output values are shown below.

Following are decimal outputs for unusual input numbers:

Input number	DEC_SIGN	Decimal digits buffer
NaN	.	. followed by blanks
+ infinity	+	+ followed by blanks
− infinity	−	− followed by blanks
+0	0	blanks
−0	−	0 followed by blanks

Function

mqcBIN_DECLOW converts the floating-point number of type BIN_TYPE, located at BIN_PTR, into a decimal representation stored in DEC_PTR, DEC_EXPONENT, and SIGN. The caller uses these quantities to construct the final output format of the number.

mqcBIN_DECLOW provides only 18 decimal digits. If DEC_LENGTH is greater than 18, the DEC_PTR buffer is filled with 18 decimal digits, followed by ASCII underscore (5F hex) characters. The decimal point is assumed to be to the right of the rightmost underscore. Thus, the underscore can be interpreted as "unknown decimal digit."

Errors

mqcBIN_DECLOW has three possible errors:

1. Invalid input. This error occurs only when the input DEC_LENGTH is given as 0 and the input number is −0, any infinity, or NaN. In these cases, the empty decimal digits buffer that the caller requested would have contained information other than decimal digits. The DEC_SIGN and DEC_EXPONENT outputs are set appropriately. If the 80287 I exception is unmasked, the trap handler is called with the input number on the 80287 stack.
2. Denormalized input. This error occurs only when BIN_TYPE is TEMP_REAL and the leading significand bit is zero. DEC_SIGN and DEC_EXPONENT are given the correct output values. The output decimal digit string is given leading zeroes to indicate the extent of denormalization. If the 80287 D exception is unmasked, the trap handler is called with the input number on the 80287 stack.
3. Inexact result. The correct answer is output. If the 80287 P exception is unmasked, the trap handler is called with the input number on the 80287 stack.

Example of PL/M-286 Use

```
mqcBIN_DECLOW: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcBIN_DECLOW;

DECLARE REAL_VAR3 REAL;
DECLARE DIGITS_BUFFER(6) BYTE;
DECLARE FINAL_BUFFER(15) BYTE;

/* Assume that REAL_VAR3 contains the hex value
   C0E9A36D. */

BIN_DECLOW_BLOCK.BIN_PTR = @REAL_VAR3;
BIN_DECLOW_BLOCK.BIN_TYPE = SHORT_REAL;
BIN_DECLOW_BLOCK.DEC_LENGTH = 6;
BIN_DECLOW_BLOCK.DEC_PTR = @DIGITS_BUFFER;
CALL mqcBIN_DECLOW(@BIN_DECLOW_BLOCK);

/* The "building blocks" returned by mqcBIN_DECLOW are
   assembled into a six-digit scientific notation
   format */

FINAL_BUFFER(0) = BIN_DECLOW_BLOCK.DEC_SIGN;
FINAL_BUFFER(1) = DIGITS_BUFFER(0);
FINAL_BUFFER(2) = '.';
CALL MOVB( @DIGITS_BUFFER(1), @FINAL_BUFFER(3), 5 );
FINAL_BUFFER(8) = 'E';

/* Assume DECIMAL_WORD_OUT translates the first INTEGER
   parameter into a string of decimal digits placed at the
   second POINTER parameter */

CALL DECIMAL_WORD_OUT (BIN_DECLOW_BLOCK.DEC_EXPONENT + 5,
@FINAL_BUFFER(9));

/* FINAL_BUFFER now contains the string "-7.30120E0" */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcBIN_DECLOW: FAR

REAL_VAR3           DD              0C0E9A36DR
DIGITS_BUFFER_3     DB              6 DUP (?)
FINAL_BUFFER        DB              15 DUP (?)
BLOCK_3             BIN_DECLOW_BLOCK < REAL_VAR3,
&                     SHORT_REAL,?,DIGITS_BUFFER_3,?,? >

     MOV BLOCK_3.DEC_LENGTH, 6    ; plug in the correct
                                  ; length
     MOV DX, SEG(BLOCK_3)         ; upper part of param
                                  ; address
     PUSH DX                      ; goes onto stack
     MOV DX, OFFSET(BLOCK_3)      ; lower part of param
                                  ; address
     PUSH DX                      ; goes onto stack
     CALL mqcBIN_DECLOW

; Now DIGITS_BUFFER contains the string "730120", DEC_SIGN
; equals "-", and DEC_EXPONENT equals -5. This
; information can be used to construct a number in any
; format desired.
```

DEC_BIN — mqcDEC_BIN (Convert decimal string to binary number)

Parameter

DEC_BIN_BLOCK_PTR, a double-word that points to a ten-byte block of memory. The block of memory contains four fields that hold the following variables:

- BIN_PTR, a double-word that points to the output binary number. The amount of memory space occupied by the number is determined by BIN_TYPE.
- BIN_TYPE, a one-byte input code that indicates the type (SHORT_REAL, LONG_REAL, or TEMP_REAL) of the number pointed to by BIN_PTR. The codes for the types are 0, 2, and 3, respectively. The caller must ensure that BIN_TYPE is set to one of the indicated legal values; any other value gives an undefined result.
- DEC_LENGTH, a one-byte input that indicates the length of the memory field pointed to by DEC_PTR. The length must be between 1 and 255; 0 will give an undefined result.
- DEC_PTR, a double-word that points to the input string of ASCII decimal digits. The format of the input string is given earlier in this chapter. The caller should already have verified correct input syntax before calling mqcDEC_BIN.

Function

mqcDEC_BIN converts the string pointed to by DEC_PTR into a binary floating-point number of type BIN_TYPE and leaves the number at the memory location pointed to by BIN_PTR.

Errors

mqcDEC_BIN has six possible errors:

1. Illegal input. If the input string is not of the specified format, the output is undefined. No warning is issued.
2. Overflow beyond the TEMP_REAL format. If the 80287 O exception is masked, a correctly signed infinity is stored in the output buffer. If it is not masked, the 80287 trap handler is called with INDEFINITE on the 80287 stack, and nothing is written to the output buffer.
3. Overflow beyond the output format, but not the TEMP_REAL format. If the 80287 O exception is masked, the correctly signed infinity is stored in the output buffer. If the 80287 O exception is unmasked, the trap handler is called. In this case, the top of the 80287 stack contains the output number with the significand rounded to the output format's precision.
4. Underflow of the TEMP_REAL format. If the 80287 U exception is masked, the output buffer contains the result of gradual underflow. If U is unmasked, INDEFINITE is left on the 80287 stack, the output buffer is unchanged, and the 80287 trap handler is called.
5. Underflow of the output format, but not of the TEMP_REAL format. The output buffer contains the result of gradual underflow. If the 80287 U exception is unmasked, the trap handler is called. In this case, the top of the 80287 stack contains the output number with the significand rounded to the output format's precision.
6. An inexact result. The output buffer contains the result rounded to the output format. If the 80287 P exception is unmasked, the trap handler is called with the TEMP_REAL result on the 80287 stack. Because this condition is not usually considered an error, the P exception is usually masked and the P error bit is ignored.

Example of PL/M-286 Use

```
mqcDEC_BIN: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcDEC_BIN;

DECLARE REAL_VAR REAL;
DECLARE INPUT_BUFFER (120) BYTE;
DECLARE ACTUAL BYTE;

/* Assume that at this point a decimal string (e.g.
   "-3.4E2") has already been placed into INPUT_BUFFER,
   and a buffer length (6 for the above number) has been
   placed into ACTUAL */

DEC_BIN_BLOCK.BIN_PTR = @REAL_VAR;
DEC_BIN_BLOCK.BIN_TYPE = SHORT_REAL;
DEC_BIN_BLOCK.DEC_LENGTH = ACTUAL;
DEC_BIN_BLOCK.DEC_PTR = @INPUT_BUFFER;
CALL mqcDEC_BIN (@DEC_BIN_BLOCK);

/* REAL_VAR now equals the encoded SHORT_REAL value. For
   the sample string given, REAL_VAR = -340 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs: EXTRN mqcDEC_BIN: FAR

INPUT_BUFFER DB           100 DUP (?)
ACTUAL       DW           0
REAL_VAR     DQ           0
BLOCK_1      DEC_BIN_BLOCK  < REAL_VAR, LONG_REAL, ?,
&              INPUT_BUFFER >

ENTRY:
  ; Assume that at this point a decimal string (e.g.
  ; "-3.4E2") has already been placed into INPUT_BUFFER,
  ; and a buffer length (6 for the above number) has been
  ; placed into ACTUAL.

     MOV AX, ACTUAL                  ; get length of string
     MOV BLOCK_1.DEC_LENGTH, AL      ; place byte into param
                                     ; block
     MOV DX, SEG(BLOCK_1)            ; top half of param
                                     ; block ptr
     PUSH DX                         ; goes onto stack
     MOV DX, OFFSET(BLOCK_1)         ; bottom half of param
                                     ; block ptr
     PUSH DX                         ; goes onto stack
     CALL mqcDEC_BIN                 ; convert string to
                                     ; binary

; REAL_VAR now equals the encoded LONG_REAL value. For
; the sample given, REAL_VAR = -340.
```

DECLOW_BIN — mqcDECLOW_BIN (Convert decimal string, low-level interface, to binary number)

Parameter

DECLOW_BIN_BLOCK_PTR, a double-word that points to a 13-byte block of memory. The block of memory contains six fields that hold the following variables:

- BIN_PTR, a double-word that points to the output binary number. The amount of memory space occupied by the number is determined by BIN_TYPE.
- BIN_TYPE, a one-byte input code that indicates the type (SHORT_REAL, LONG_REAL, or TEMP_REAL) of the number pointed to by BIN_PTR. The codes for the types are 0, 2, and 3, respectively. The caller must ensure that BIN_TYPE is set to one of the indicated legal values; any other value gives an undefined result.
- DEC_LENGTH, a one-byte input that indicates the length of the memory field pointed to by DEC_PTR. The length must be between 1 and 21, inclusive. Any other value gives an undefined result. If DEC_LENGTH equals 21, the original significand has more than 20 digits. In this case, the given significand consists of the first 20 digits only. The twenty-first digit is the least significant nonzero digit of the original significand.
- DEC_PTR, a double-word that points to the input string of ASCII decimal digits. The digits give the significand of the number to be converted. Leading zeroes, the sign, the decimal point, the exponent, and lagging zeroes have already been removed.
- DEC_EXPONENT, a word that gives the base-10 exponent of the number to be converted. It is assumed that the decimal point has been shifted to the immediate right of the last digit of the significand. DEC_EXPONENT is then the power of ten, which, when multiplied by the significand, yields the number to be converted. Negative powers are represented in two's complement form.
- DEC_SIGN, a byte that gives the sign of the number to be converted. The possible values are ASCII "+" (2B hex) and ASCII "−" (2D hex).

Function

mqcDECLOW_BIN accepts a decimal number that has already been converted from a higher-level format to the lower-level format described under DEC_PTR, DEC_EXPONENT, and DEC_SIGN above. It converts the number into a binary floating-point number of type BIN_TYPE and leaves the number at the memory location pointed to by BIN_PTR.

Errors

All errors (invalid input, overflow, underflow, and inexact result) are the same as for mqcDEC_BIN and have the same consequences.

Example of PL/M-286 Use

```
mqcDECLOW_BIN: PROCEDURE (BLOCK_PTR) EXTERNAL;
  DECLARE BLOCK_PTR POINTER;
END mqcDECLOW_BIN;

DECLARE REAL_VAR2 REAL;
DECLARE N_DIGITS BYTE;
DECLARE DIGITS_BUFFER(21) BYTE;
```

```
/* Assume that at this point the string "371" has been
   placed into DIGITS_BUFFER, and N_DIGITS has been set to
   the string length 3 */

/* The following code assumes DIGIT_BUFFER contains a
   decimal string representing an integer number of
   pennies. It places the number of dollars, in SHORT_REAL
   binary format, into REAL_VAR2. */

BIN_DECLOW_BLOCK.BIN_PTR = @REAL_VAR2;
BIN_DECLOW_BLOCK.BIN_TYPE = SHORT_REAL;
BIN_DECLOW_BLOCK.DEC_LENGTH = N_DIGITS;
BIN_DECLOW_BLOCK.DEC_PTR = @DIGITS_BUFFER;
BIN_DECLOW_BLOCK.DEC_EXPONENT = -2;
BIN_DECLOW_BLOCK.DEC_SIGN = '+';
CALL mqcDECLOW_BIN(@BIN_DECLOW_BLOCK);

/* REAL_VAR2 now equals the encoded SHORT_REAL value
   3.71 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcDECLOW_BIN: FAR

N_DIGITS          DB                  0
DIGITS_BUFFER     DB                  21 DUP (?)
REAL_VAR2         DQ                  ?
BLOCK_2           DECLOW_BIN_BLOCK < REAL_VAR2,
&                   LONG_REAL,?,DIGITS_BUFFER,-2,'+'>

; assume that the string "371" has been placed into
; DIGITS_BUFFER, and N_DIGITS has been set to the string
; length 3.

        MOV AL,N_DIGITS                 ; load string length
        MOV BLOCK_2.DEC_LENGTH, AL      ; place into param
                                        ; block
        MOV DX, SEG(BLOCK_2)            ; top half of param
                                        ; block ptr
        PUSH DX                         ; goes onto stack
        MOV DX, OFFSET(BLOCK_2)         ; bottom half of
                                        ; param block ptr
        PUSH DX                         ; goes onto stack
        CALL mqcDECLOW_BIN              ; convert string to
                                        ; binary

; The encoded LONG_REAL value 3.71 has now been placed
; into REAL_VAR2.
```

LONG_TEMP — mqcLONG_TEMP (Convert LONG_REAL to TEMP_REAL)

Parameters

LONG_REAL_PTR points to the eight bytes of memory that hold the input LONG_REAL binary floating-point number.

TEMP_REAL_PTR points to the ten bytes of memory into which the TEMP_REAL answer is placed.

Function

The input LONG_REAL number is extended to the TEMP_REAL number that represents the same value. Unusual inputs (NaNs, infinities, −0) translate to the same unusual outputs, with zeroes padding the end of the output significand.

Errors

The only possible error is a denormalized input value. The corresponding denormalized output value is left in the output buffer. If the 80287 D exception is unmasked, the trap handler is called with the output on the 80287 stack.

Example of PL/M-286 Use

```
mqcLONG_TEMP: PROCEDURE (LONG_REAL_PTR,TEMP_REAL_PTR)
              EXTERNAL;
  DECLARE (LONG_REAL_PTR,TEMP_REAL_PTR) POINTER;
END mqcLONG_TEMP;

DECLARE TEMP_REAL_ARRAY (8) STRUCTURE
       (SIGNIFICAND(8) BYTE, EXPO_SIGN(2) INTEGER);
DECLARE LONG_REAL_ARRAY (8) STRUCTURE (NUM(8) BYTE);
DECLARE I BYTE;

/* The following code expands an array of LONG_REAL
   variables into an array of TEMP_REAL variables */

  DO I = 0 TO 7;
   CALL mqcLONG_TEMP ( @LONG_REAL_ARRAY(I),
   @TEMP_REAL_ARRAY(I) );
  END;
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcLONG_TEMP: FAR

TEMP_NUM  DT  ?
LONG_NUM  DQ  ?

; The following code transfers the LONG_REAL number at
; LONG_NUM to the TEMP_REAL number at TEMP_NUM

    MOV DX, SEG(LONG_NUM)        ; top half of parameter
```

```
                                  ; pointer 1
PUSH DX                           ; goes onto stack
MOV DX, OFFSET(LONG_NUM)          ; bottom half of
                                  ; parameter pointer 1
PUSH DX                           ; goes onto stack
MOV DX, SEG(TEMP_NUM)             ; top half of parameter
                                  ; pointer 2
PUSH DX                           ; goes onto stack
MOV DX, OFFSET(TEMP_NUM)          ; bottom half of
                                  ; parameter pointer 2
PUSH DX                           ; goes onto stack
CALL mqcLONG_TEMP
```

SHORT_TEMP — mqcSHORT_TEMP (Convert SHORT_REAL to TEMP_REAL)

Parameters

SHORT_REAL_PTR points to the four bytes of memory that hold the input SHORT_REAL binary floating-point number.

TEMP_REAL_PTR points to the ten bytes of memory into which the TEMP_REAL answer is placed.

Function

The input SHORT_REAL number is extended to the TEMP_REAL number, which represents the same value. Unusual inputs (NaNs, infinities, −0) translate to the same unusual outputs, with zeroes padding the end of the output significand.

Errors

The only possible error is a denormalized input value. The corresponding denormalized output value is left in the output buffer. If the 80287 D exception is unmasked, the trap handler is called with the output on the 80287 stack.

Example of PL/M-286 Use

```
mqcSHORT_TEMP: PROCEDURE (SHORT_REAL_PTR,TEMP_REAL_PTR)
                EXTERNAL;
   DECLARE (SHORT_REAL_PTR,TEMP_REAL_PTR) POINTER;
END mqcSHORT_TEMP;

DECLARE TEMP_REAL_ARRAY (8) STRUCTURE (NUM(10) BYTE);
DECLARE SHORT_REAL_ARRAY (8) REAL;
DECLARE I BYTE;

/* The following loop expands an array of SHORT_REAL
   variables into an array of TEMP_REAL variables. */

   DO I = 0 TO 7;
     CALL mqcSHORT_TEMP ( @SHORT_REAL_ARRAY(I),
     @TEMP_REAL_ARRAY(I) );
   END;
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcSHORT_TEMP: FAR

SHORT_X    DD   ?
TEMP_X     DT   ?

; The following code converts the SHORT_REAL number at
; SHORT_X into a TEMP_REAL value, and leaves it at TEMP_X.

; It is assumed that both numbers are in the DS segment.
```

```
PUSH DS                      ; top half of first
                             ; parameter
MOV AX, OFFSET(SHORT_X)      ; bottom half of first
                             ; parameter
PUSH AX                      ; pushed onto stack
PUSH DS                      ; top half of second
                             ; parameter
MOV AX, OFFSET(TEMP_X)       ; bottom half of second
                             ; parameter
PUSH AX                      ; pushed onto stack
CALL mqcSHORT_TEMP           ; conversion is now
                             ; complete
```

TEMP_LONG — mqcTEMP_LONG (Convert TEMP_REAL to LONG_REAL)

Parameters

TEMP_REAL_PTR points to the ten bytes of memory that hold the input TEMP_REAL binary floating-point number.

LONG_REAL_PTR points to the eight bytes of memory into which the LONG_REAL answer is placed.

Function

The TEMP_REAL number is rounded to the nearest LONG_REAL number. If two LONG_REAL numbers are equally near, the one with a zero least significant bit is chosen.

Errors

mqcTEMP_LONG has five possible errors:

1. Overflow. If the exponent of the TEMP_REAL input exceeds the output capacity, the correctly signed infinity is placed in the output buffer. If the 80287 O exception is unmasked, the trap handler is called. In this case, the number left on the 80287 stack is the input number, with the exponent unchanged and the significand rounded to the output's precision.
2. Underflow. The output buffer is filled with the result of gradual underflow, rounded to fit the output format. If the 80287 U exception is unmasked, the trap handler is called. In this case, the 80287 stack is left with the input number's significand rounded to the output precision, with the too-small exponent unchanged.
3. Inexact. This occurs when the input significand has nonzero bits outside the field of the output format. The correctly rounded answer is left in the output buffer. If the 80287 P exception is unmasked, the trap handler is called with the input number on the 80287 stack.
4. Unnormalized invalid input. This occurs when the input is unnormalized, no underflow occurs, and rounding to the output precision does not normalize the answer. INDEFINITE is left in the output buffer. If the 80287 I exception is unmasked, the trap handler is called with the input number on the 80287 stack.
5. Invalid nonzero NaN truncation. The only NaN inputs faithfully represented in the smaller format are those for which the significand bits being truncated are all zero. For other NaNs, invalid operation is signalled. The output buffer is filled with the truncated result only if the truncated significand is nonzero. If the significand is zero, it is made nonzero by substituting the highest nonzero byte of the original significand into bits 5 through 12 of the new LONG_REAL significand. If the 80287 I exception is unmasked, the trap handler is called with the input number on the 80287 stack.

Example of PL/M-286 Use

```
mqcTEMP_LONG: PROCEDURE (TEMP_REAL_PTR,LONG_REAL_PTR)
              EXTERNAL;
  DECLARE (TEMP_REAL_PTR,LONG_REAL_PTR) POINTER;
END mqcTEMP_LONG;
DECLARE TEMP_REAL_ARRAY (8) STRUCTURE (NUM(10) BYTE);
DECLARE LONG_REAL_ARRAY (8) STRUCTURE (NUM(8) BYTE);
```

```
DECLARE I BYTE;

/* The following code transfers an array of 8 TEMP_REAL
   numbers to an array of LONG_REAL numbers. */

 DO I = 0 TO 7;
  CALL mqcTEMP_LONG(@TEMP_REAL_ARRAY(I),
  @LONG_REAL_ARRAY(I)) ;
 END;
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcTEMP_LONG: FAR

TEMP_REAL_ARRAY    DT    8 DUP (?)
LONG_REAL_ARRAY    DQ    8 DUP (?)

; The following code transfers the array of 8 TEMP_REAL
; numbers in TEMP_REAL_ARRAY to the 8 LONG_REAL numbers in
; LONG_REAL_ARRAY.

; It is assumed that both arrays are in the current DS
; segment.

     PUSH BP                            ; BP should always
                                        ; be preserved
     MOV BP, OFFSET(LONG_REAL_ARRAY)
     MOV DX, OFFSET(TEMP_REAL_ARRAY)    ; DX will not be
                                        ; preserved
     MOV CX, 8                          ; initialize loop
                                        ; counter
LOOP8:
     PUSH CX                            ; preserve counter
     PUSH DX                            ; preserve TEMP_REAL
                                        ; pointer
     PUSH DS                            ; top half of TEMP
                                        ; parameter
     PUSH DX                            ; bottom half of
                                        ; TEMP parameter
     PUSH DS                            ; top half of LONG
                                        ; parameter
     PUSH BP                            ; bottom half of
                                        ; LONG parameter
     CALL mqcTEMP_LONG                  ; perform conversion
     POP DX                             ; restore TEMP_REAL
                                        ; pointer
     ADD DX, 10                         ; increment to next
                                        ; TEMP_REAL
     ADD BP, 8                          ; increment to next
                                        ; LONG_REAL
     POP CX                             ; restore counter
     DEC CX                             ; count down
     JNZ LOOP8                          ; loop back if more
                                        ; conversions
     POP BP                             ; restore the
                                        ; original BP
```

TEMP_SHORT — mqcTEMP_SHORT (Convert TEMP_REAL to SHORT_REAL)

Parameters

TEMP_REAL_PTR points to the ten bytes of memory that hold the input TEMP_REAL binary floating-point number.

SHORT_REAL_PTR points to the four bytes of memory into which the SHORT_REAL answer is placed.

Function

The TEMP_REAL number is rounded to the nearest SHORT_REAL number. If two SHORT_REAL numbers are equally near, the one with a zero least significant bit is chosen.

Errors

The errors (overflow, underflow, inexact, invalid unnormalized, and invalid nonzero NaN truncation) are the same as for mqcTEMP_LONG, with the same actions taken except for the subcase of error 5—in which the highest nonzero byte of the original significand must be copied to the result significand to preserve its character as a NaN. In this case, that byte occupies bits 0 through 7 of the new SHORT_REAL significand.

Example of PL/M-286 Use

```
mqcTEMP_SHORT: PROCEDURE (TEMP_REAL_PTR,SHORT_REAL_PTR)
               EXTERNAL;
  DECLARE (TEMP_REAL_PTR,SHORT_REAL_PTR) POINTER;
END mqcTEMP_SHORT;

DECLARE TEMP_REAL_ARRAY (8) STRUCTURE
        (SIGNIFICAND(8) BYTE, EXPO_SIGN(2) INTEGER);
DECLARE SHORT_REAL_ARRAY (8) REAL;
DECLARE I BYTE;

/* The following loop transfers an array of 8 TEMP_REAL
   numbers (which could, for example, represent an image
   of the 80287 stack) to a more compact array of
   SHORT_REAL numbers. */

 DO I = 0 TO 7;
   CALL mqcTEMP_SHORT ( @TEMP_REAL_ARRAY(I),
   @SHORT_REAL_ARRAY(I) );
END;
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqcTEMP_SHORT: FAR

TEMP_VAR  DT  ?
SHORT_VAR DD  ?
```

```
; The following code transfers the TEMP_REAL number at
; TEMP_VAR to the SHORT_REAL number at SHORT_VAR

    MOV DX, SEG(TEMP_VAR)          ; top half of parameter
                                   ; pointer
    PUSH DX                        ; goes onto stack
    MOV DX, OFFSET(TEMP_VAR)       ; bottom half of
                                   ; parameter pointer
    PUSH DX                        ; goes onto stack
    MOV DX, SEG(SHORT_VAR)         ; top half of parameter
                                   ; pointer
    PUSH DX                        ; goes onto stack
    MOV DX, OFFSET(SHORT_VAR)      ; bottom half of
                                   ; parameter pointer
    PUSH DX                        ; goes onto stack
    CALL mqcTEMP_SHORT
```

Binding DC287.LIB to Program Modules

The final action you must take to use DC287.LIB in your programs is to insert the file name DC287.LIB into the BND286 command. You must also bind in 80287.LIB.

Following is the suggested order for the object modules in your BND286 statement:

Your object modules
DC287.LIB
CEL287.LIB (if you are using it)
EH287.LIB (if you are using it)
80287.LIB

For example, if you are binding your PL/M-286 modules MYMOD1.OBJ and MYMOD2.OBJ into a program, issue the following command:

```
-BND286 :F1:MYMOD1.OBJ, &<cr>
        :F1:MYMOD2, &<cr>
        :F0:DC287.LIB, &<cr>
        :F0:80287.LIB &<cr>
        OBJECT (:F1:MYPROG.LNK)<cr>
```

If you have a single ASM286-generated object module :F1:MYPROG.OBJ to be executed, issue the following command:

```
-BND286 :F1:MYPROG.OBJ, &<cr>
        :F0:DC287.LIB, &<cr>
        :F0:80287.LIB &<cr>
        OBJECT (:F1:MYPROG.LNK)<cr>
```

Overview

This chapter describes the functions of CEL287.LIB, a library that contains commonly-used functions of floating-point arithmetic.

The 80287 chip gives direct support for the most time-consuming part of the computation of every CEL287 function, but the forms of the functions and the ranges of the inputs are limited. CEL287 provides the software support necessary to let the 80287 provide a complete package of elementary functions, giving valid results for all appropriate inputs.

All CEL287 procedure names begin with the four letters *mqer*. This prefix has been added to reduce the possibility of conflict between a CEL287 name and another name in your program. The *mqer* prefix appears in lowercase letters throughout this chapter to make the names more readable.

Following is a summary of CEL287 grouped by functions.

Rounding and Truncation Functions

- mqerIEX rounds a real number to the nearest integer; to the even integer if a tie occurs. The answer returned is real.
- mqerIE2 is as mqerIEX, except the answer is a 16-bit integer.
- mqerIE4 is as mqerIEX, except the answer is a 32-bit integer.
- mqerIAX rounds a real number to the nearest integer; to the integer away from zero if a tie occurs. The answer returned is real.
- mqerIA2 is as mqerIAX, except the answer is a 16-bit integer.
- mqerIA4 is as mqerIAX, except the answer is a 32-bit integer.
- mqerICX chops the fractional part of a real input. The answer is real.
- mqerIC2 is as mqerICX, except the answer is a 16-bit integer.
- mqerIC4 is as mqerICX, except the answer is a 32-bit integer.

Logarithmic and Exponential Functions

- mqerLGD computes decimal (base 10) logarithms.
- mqerLGE computes natural (base e) logarithms.
- mqerEXP computes exponentials of base e.
- mqerY2X computes exponentials of any base.
- mqerYI2 computes an input real to the AX integer power.
- mqerYI4 is as mqerYI2, except the input integer is DXAX.
- mqerYIS is as mqerYI2, except the input integer is on the 80286 stack.

Trigonometric and Hyperbolic Functions

- mqerSIN, mqerCOS, mqerTAN compute sine, cosine, and tangent.
- mqerASN, mqerACS, mqerATN compute the corresponding inverse functions.

- mqerSNH, mqerCSH, mqerTNH compute the corresponding hyperbolic functions.
- mqerAT2 is a special version of the arc tangent function. It handles all the quadrants for rectangular-to-polar conversion in two dimensions.

Other Functions

- mqerDIM is FORTRAN's positive difference function.
- mqerMAX computes the maximum of two real inputs.
- mqerMIN computes the minimum of two real inputs.
- mqerSGH combines the sign of one input with the magnitude of the other input.
- mqerMOD computes a modulus, retaining the sign of the dividend.
- mqerRMD computes a modulus, giving the value closest to zero.

Declaring CEL287 Procedures in ASM286 Programs

In each source program module that calls a CEL287 procedure, you must declare that procedure as external before it is used.

Following are declarations you should make at the top of your ASM286 program. Include only those procedures you use.

For easier referencing, the EXTRN statements in the examples for the individual functions have been duplicated. You should not duplicate the EXTRN statements in your program.

All CEL287 procedures must be declared FAR.

```
; All of the following EXTRN statements should appear
; outside of all SEGMENT-ENDS pairs

EXTRN mqerACS: FAR
EXTRN mqerASN: FAR
EXTRN mqerAT2: FAR
EXTRN mqerATN: FAR
EXTRN mqerCOS: FAR
EXTRN mqerCSH: FAR
EXTRN mqerDIM: FAR
EXTRN mqerEXP: FAR
EXTRN mqerIA2: FAR
EXTRN mqerIA4: FAR
EXTRN mqerIAX: FAR
EXTRN mqerIC2: FAR
EXTRN mqerIC4: FAR
EXTRN mqerICX: FAR
EXTRN mqerIE2: FAR
EXTRN mqerIE4: FAR
EXTRN mqerIEX: FAR
EXTRN mqerLGD: FAR
EXTRN mqerLGE: FAR
EXTRN mqerMAX: FAR
EXTRN mqerMIN: FAR
EXTRN mqerMOD: FAR
EXTRN mqerRMD: FAR
EXTRN mqerSGN: FAR
EXTRN mqerSIN: FAR
```

```
EXTRN mqerSNH: FAR
EXTRN mqerTAN: FAR
EXTRN mqerTNH: FAR
EXTRN mqerY2X: FAR
EXTRN mqerYI2: FAR
EXTRN mqerYI4: FAR
EXTRN mqerYIS: FAR
```

Declaring CEL287 Procedures in PL/M-286 Programs

Following are the declarations you should make at the top of your PL/M-286 programs. Include only those procedures you use.

For easier referencing, the procedure declarations in the examples for the individual functions have been duplicated. You should not duplicate the declarations in your programs.

To use CEL287.LIB, you must compile your PL/M-286 modules under the MEDIUM or LARGE computation models, because CEL287's procedures are FAR procedures.

```
mqerACS: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerACS;

mqerASN: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerASN;

mqerAT2: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerAT2;

mqerATN: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerATN;

mqerCOS: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerCOS;

mqerCSH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerCSH;

mqerDIM: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerDIM;

mqerEXP: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerEXP;

mqerIA2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIA2;
```

```
mqerIAX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerIAX;

mqerIC2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIC2;

mqerICX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerICX;

mqerIE2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIE2;

mqerIEX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerIEX;

mqerLGD: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerLGD;

mqerLGE: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerLGE;

mqerMAX: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMAX;

mqerMIN: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMIN;

mqerMOD: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMOD;

mqerRMD: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerRMD;

mqerSGN: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerSGN;

mqerSIN: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerSIN;

mqerSNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerSNH;

mqerTAN: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTAN;
```

```
mqerTNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTNH;

mqerY2X: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerY2X;

mqerYIS: PROCEDURE (Y,I) REAL EXTERNAL;
  DECLARE Y REAL, I INTEGER;
END mqerYIS;
```

Calling CEL287 Functions in PL/M-286 Programs

CEL287 functions operate with numbers on the 80287 stack; however, this fact is invisible to you when you are programming in PL/M-286. No PL/M statements explicitly load and unload numbers from the 80287 stack. The code for this is implicitly generated whenever arithmetic involving REAL numbers is programmed.

To invoke a CEL287 function after you have declared it, simply use the function in an assignment statement (or in an expression in any other context). All CEL287 functions, if declared as shown in the examples, will cause the PL/M compiler to load the input parameters when CEL287 is invoked and to place the answer in the correct destination after the CEL287 procedure is complete.

PL/M-286 does not support the LONG_REAL and TEMP_REAL formats. Numeric constants and variables used by PL/M are 32-bit SHORT_REAL numbers, in 80286 memory, that are converted to TEMP_REAL when placed on the 80287 stack. When PL/M stores the result of a CEL287 function that returns a value on the 80287 stack, the function is converted from TEMP_REAL to SHORT_REAL. The conversion takes place with an FSTP instruction generated by the compiler.

The FSTP instruction may possibly generate an O overflow or a U underflow error. (Indeed, because TEMP_REAL is such a large format, this is where O and U errors are most likely to occur.) These errors are beyond the control of the CEL287 function; hence, the outlines in this chapter that cover the values placed into 80287 registers do not apply for these errors.

Consult the *iAPX 286 Programmer's Reference Manual Numeric Supplement* for specifications of the FSTP instruction.

Stack Use of CEL287 Procedures

CEL287 requires 50 bytes of the 80286 stack for its internal storage. If, however, CEL287 is interrupted by a program that itself calls CEL287 (for example, an error trap handler), an additional 50 bytes of 80286 stack are needed for each recursion level possible. CEL287 declares a 50-byte STACK segment within its program modules, automatically allocating the first 50 bytes by BND286 or BLD286.

CEL287 requires four stack positions on the 80287 stack. Input parameters are counted in the four numbers. This means that if a CEL287 function has input parameters at the top two stack positions, ST(0) and ST(1), the parameters will be destroyed by the function. Also, the numbers ST(6) and ST(7), must be empty for the function to execute correctly. If the 80287 stack overflows during a CEL287 function, the results are undefined.

All CEL287 functions are reentrant in that they use no fixed 80286 memory locations to store internal variables. However, since the 80287 stack contains only eight elements, it is advisable for an interrupt routine to explicitly preserve the 80287 stack upon entry and to restore it upon exit. Any procedure that interrupts a CEL287 function must save the entire 80287 stack, because it is impossible to tell which four stack elements are being used by CEL287.

Register Use of CEL287 Procedures

CEL287 conforms to the register-saving conventions of all 80286 high level languages provided by Intel.

According to these conventions, the 80286 registers that can be destroyed by CEL287 functions are AX, BX, CX, DX, DI, SI, and ES. The 80286 registers that are unchanged by CEL287 functions are BP, SS, and DS. The SP register is changed only by mqerYIS, which accepts a parameter on the 80286 stack and returns with the parameter popped from the stack.

All CEL287 functions save the 80287 Control Word upon entry, and restore it upon return. During computation, all CEL287 functions except mqerDIM use TEMP_REAL precision, and a rounding mode of CEL287's choosing. mqerDIM uses the precision and rounding in effect at the time mqerDIM is called.

The 80287 exception flags (error bits) are treated by all CEL287 functions as "sticky." This means that the only change to error bits that CEL287 may make is from off to on. If an error bit is already on when CEL287 is called, it will stay on. This allows you to perform a sequence of 80287 calculations that involve CEL287 functions with some of the 80287 errors masked. At the end of the calculation, if the exception flag for a masked error is still zero, you can be sure that the error was never encountered.

Error Reporting in CEL287.LIB

CEL287 incorporates the 80287 error-reporting mechanism. Whenever an error is detected, an appropriate 80287 exception bit is set. Using 80287 instructions, you can set your choice of unmasked bits and provide procedures that will be called whenever an unmasked exception is detected. This way, you do not have to test for errors every time you invoke a CEL287 function.

There are only seven CEL287 functions that leave a meaningful value in the P precision error bit. These are mqerIC2, mqerIC4, mqerICX, mqerIE2, mqerIE4, mqerIEX, and mqerSGN. For all other CEL287 functions, the P exception should be masked, and the P exception bit returned should be ignored.

Whenever an unmasked exception is detected by any CEL287 function, the 80287 calls the trap handler. The values on the 80287 stack are decribed under each individual CEL287 function. In addition, CEL287 sets the following values whenever the trap handler is called:

- The return address from the CEL287 function, consisting of the selector-offset pair, is placed in the memory operand pointer of the 80287.
- The 80287 control word is restored to the value it had when the CEL287 function was called.
- The 80287 instruction pointer offset is set to 0FFFFF hex.
- The 80287 opcode register is set to the code described under each CEL287 function. The description gives the code in the form of three hexadecimal digits.

The first (most significant) digit gives the number of values remaining on the 80287 stack that relate to the CEL287 function that caused the trap. The last two digits identify the CEL287 function.

The trap handler can identify an interrupt as generated by CEL287 by examining the 80287 instruction offset to see that it is set to 0FFFFF hex. If it is, then the identity of the CEL287 function is determined by examining the selector portion of the 80287 instruction pointer.

ACS — mqerACS x = arc cosine(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerACS returns the number, in radians, whose cosine is equal to the input value. Results are in the range 0 to π. Valid inputs are from -1 to 1. All zeroes, pseudo-zeroes, and denormals return the value $\pi/2$. Also, unnormals less than 2^{-63} return the value $\pi/2$.

Output

The answer replaces the input on the 80287 stack.

Errors

Because cosines exist only in the range -1 to 1, all inputs to mqerACS outside of this range are invalid. Thus, an I error is given for infinities, NaNs, values less than -1, and values greater than 1. An I error is given also for unnormals not less than 2^{-63}.

If I is unmasked, the trap handler is called with the input still on the stack, and the 80287 opcode register set to 175 hex. If I is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

Example of PL/M-286 Use

```
mqerACS: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerACS;

DECLARE HYPOTENUSE REAL;    /* Longest side of a right
                               triangle */
DECLARE ADJACENT_SIDE REAL;  /* The other side, next to
                                angle computed */
DECLARE (THETA_RADIANS, THETA_DEGREES) REAL;   /* Two forms
                                                  of the
                                                  answer */
DECLARE PI LITERALLY '3.14159265358979';

HYPOTENUSE = 10.; ADJACENT_SIDE = 5.;   /* Test values */

/* The following lines calculate the value of an angle of
   a right triangle, given the length of the hypotenuse
   and the adjacent side. mqerACS returns the value in
   radians; the value is then converted to degrees. */

THETA_RADIANS = mqerACS( ADJACENT_SIDE / HYPOTENUSE);
THETA_DEGREES = (180./PI) * THETA_RADIANS;

/* Now THETA_DEGREES = 60 -- it is a 30-60-90 degree
   triangle */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerACS: FAR

HYPOTENUSE      DQ   10.0       ; longest side of a right
                                ; triangle
ADJACENT_SIDE   DQ   5.0        ; the other side, next to
                                ; angle computed
                     ; The above initial values are test
                     ; values
THETA_RADIANS   DQ   ?
THETA_DEGREES   DQ   ?
RAD_TO_DEG      DT   4004E52EE0D31E0FBDC3R   ; the constant
                                             ; 180/PI

  ; The following lines compute the angle of a right
  ; triangle, just as in the above PL/M example, except
  ; with LONG_REAL variables.

      FLD ADJACENT_SIDE         ; (x) = ADJACENT_SIDE
      FDIV HYPOTENUSE           ; (x) = ADJACENT_SIDE/
                                ; HYPOTENUSE
      CALL mqerACS              ; angle is now in (x)
      FST THETA_RADIANS         ; radians result is saved
      FLD RAD_TO_DEG
      FMUL                      ; (x) converted to degrees
      FSTP THETA_DEGREES        ; degrees are saved, 80287
                                ; stack popped

        ; Now THETA_DEGREES = 60 -- It is a 30-60-90
        ; triangle.
```

ASN — mqerASN x = arc sine(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerASN returns the number, in radians, whose sine is equal to the input value. Results are in the range $-\pi/2$ to $\pi/2$. Valid inputs are from -1 to 1. All zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals less than 2^{-63} return the input value.

Output

The answer replaces the input on the 80287 stack.

Errors

Because sines exist only in the range -1 to 1 all inputs to mqerASN outside of this range are invalid. Thus, an I error is given for infinities, NaNs, values less than -1, and values greater than 1. An I error is also given for unnormals not less than 2^{-63}.

If I is unmasked, the trap handler is called, with the input still on the stack, and the 80287 opcode register is set to 174 hex. If I is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

Example of PL/M-286 Use

```
mqerASN: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerASN;

DECLARE HYPOTENUSE REAL;   /* Longest side of a right
                              triangle */
DECLARE OPPOSITE_SIDE REAL;   /* The other side, opposite
                                 the angle computed */
DECLARE (THETA_RADIANS, THETA_DEGREES) REAL;   /* Two forms
                                                  of the
                                                  answer */
DECLARE PI LITERALLY '3.14159265358979';

HYPOTENUSE = 10.; OPPOSITE_SIDE = 5.; /* Test values */

/* The following lines calculate the value of an angle of
   a right triangle, given the length of the hypotenuse
   and the opposite side. mqerASN returns the value in
   radians; it is then converted to degrees. */

THETA_RADIANS = mqerACS( OPPOSITE_SIDE / HYPOTENUSE);
THETA_DEGREES = (180./PI) * THETA_RADIANS;

/* Now THETA_DEGREES = 30 -- it is a 30-60-90 degree
   triangle */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerASN: FAR

HYPOTENUSE      DQ  10.0        ; longest side of a right
                                ; triangle
OPPOSITE_SIDE   DQ  5.0         ; the other side, next to
                                ; angle computed
                    ; The above initial values are test
                    ; values
THETA_DEGREES   DQ  ?
RAD_TO_DEG      DT  4004E52EE0D31E0FBDC3R   ; the constant
                                            ; 180/PI

   ; The following lines compute an angle of a right
   ; triangle, just as in the above PL/M example; except
   ; with LONG_REAL variables.

      FLD OPPOSITE_SIDE         ; (x) := OPPOSITE_SIDE
      FDIV HYPOTENUSE           ; (x) = OPPOSITE_SIDE/
                                ; HYPOTENUSE
      CALL mqerASN              ; angle is now in (x)
      FLD RAD_TO_DEG
      FMUL                      ; (x) converted to degrees
      FSTP THETA_DEGREES        ; degrees are saved, 80287
                                ; stack popped

   ; With the test inputs, THETA_DEGREES = 30; it is a
   ; 30-60-90 triangle
```

AT2 — mqerAT2 x = arc tangent(y/x)

Input Parameters

(x) is the top number on the 80287 stack; (y) is the next number on the 80287 stack.

Function

mqerAT2 performs one half of a rectangular-to-polar coordinate conversion. If the inputs (x) and (y) are interpreted as the rectangular coordinates of a point in a plane, mqerAT2 returns the angle, in radians, that the point is displaced from the positive X-axis. See figure 4-1. The angle ranges from $-\pi$ (points just below the negative X-axis) to π (points on or just above the negative X-axis).

The formulas used to calculate the angle are as follows:

If (x) > 0, return (arc tangent(y/x)).
If (x) = 0, return mqerSGN ($\pi/2$, y).
If (x) < 0, return (arc tangent(y/x)) + mqerSGN (π, y).

It is legal for one (but not both) of the inputs to be an infinity. The point is then thought of as "infinitely far out" along one of the axes, and the angle returned is the angle of that axis. For points near the X-axis, the angle retains the sign of (y). Thus, x = +INFINITY returns 0 with the sign of (y); x = −INFINITY returns π with the sign of (y); y = +INFINITY returns $+\pi/2$; y = −INFINITY returns $-\pi/2$. In all these cases, it is legal for the non-infinite input to be unnormal. Also note that the distinction between +INFINITY and −INFINITY is made even when the 80287 is in projective (unsigned infinity) mode.

mqerAT2 accepts denormal inputs. Its action depends on whether the 80287 is in normalizing mode, as indicated by the setting of the D error masking bit. If the 80287 is in normalizing mode (D is unmasked), any denormals are assumed to be zero. If the 80287 is in warning mode (D is masked), the denormals are replaced with unnormals having the same numeric value. Note that even though mqerAT2 checks the D masking bit, it does not set the D error bit; nor does it call the D trap handler.

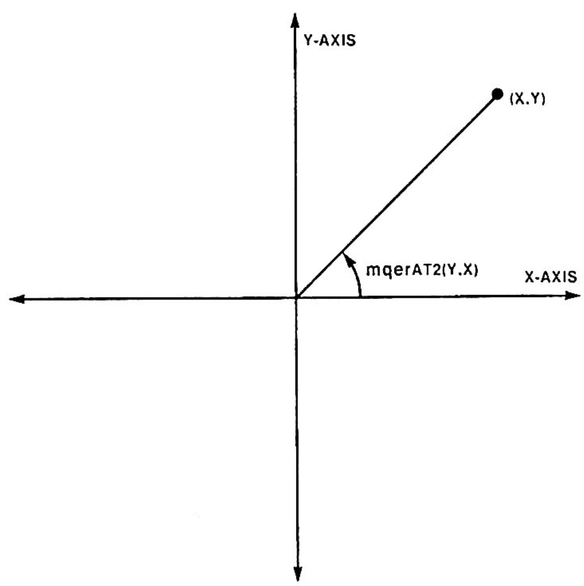

Figure 4-1. Rectangular-to-Polar Coordinate Conversion 121725-1

In some cases, mqerAT2 accepts an unnormal input with no error. This can happen only when (y) and (x) represent the coordinates of a point that is extremely close to one of the axes. A point is considered close enough to the X-axis if the absolute value of (y/x) is less than 2^{-63}. A point is considered close enough to the Y-axis if the absolute value of (x/y) is less than 2^{-63}. If the point is near the positive X-axis, the value returned is the tiny ratio (y/x) with the sign of (y). If the point is near one of the other three axes, the value returned is the angle of that axis: π with the sign of (y) for the negative X-axis, $\pi/2$ for the positive Y-axis, and $-\pi/2$ for the negative Y-axis. Under no circumstances is it legal for both inputs to be unnormal.

Output

The 80287 stack pops once, with the answer replacing the two inputs.

Errors

The first thing mqerAT2 does is check for input NaNs. If either input is a NaN, an I error exists. If I is masked, the input NaN is returned (if both inputs are NaNs, the larger NaN is returned).

An I error is given in a number of other cases. If both inputs are zero (i.e., the point on the coordinate plane is the origin), the angle cannot be defined: thus, an I error is given. If an unnormal input that does not fit any of the legal cases described above is given, an I error is given. For all these I errors, if I is masked, the value INDEFINITE is returned.

If I is unmasked for any I error, the trap handler is called with the original inputs on the 80287 stack, and the 80287 instruction selector is set to 277 hex.

If the magnitude of the result is too tiny to be represented in TEMP_REAL format, a U underflow error results. If U is masked, mqerAT2 returns a gradual underflow denormal if possible; 0 if not possible. If U is unmasked, the trap handler is called with the 80287 opcode register set to 277 hex, but the inputs are not left on the 80287 stack. Instead, the correct answer is placed on the stack with the too-small exponent presented in "wrapped" form. To obtain the true exponent, subtract decimal 24576 from the given exponent.

Example of PL/M-286 Use

```
mqerAT2: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerAT2;

FSQRT287: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END FSQRT287;

/* The above procedure is simply a PUBLIC ASM286 procedure
   which consists of the FSQRT instruction, followed by a
   RET instruction. */

DECLARE REC_X, REC_Y REAL;
DECLARE (POLAR_THETA, POLAR_R) REAL;

REC_X = 3.; REC_Y = 3. * FSQRT287(3.);    /* Test values */

/* The following statements convert the rectangular
   coordinates REC_X and REC_Y into the polar coordinates
   POLAR_R and POLAR_THETA. */
```

```
POLAR_R = FSQRT287(REC_X * REC_X + REC_Y * REC_Y);
POLAR_THETA = mqerAT2(REC_Y,REC_X);

/* Now POLAR_R = 6 and POLAR_THETA = PI/3. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerAT2: FAR

POLAR_THETA    DQ   ?
POLAR_R        DQ   ?
REC_X          DQ   3.0     ; rectangular X coordinate
                            ; initialized to test value
REC_Y          DQ   3.0     ; rectangular Y coordinate
                            ; initialized to test value

      FLD REC_Y             ; Y-coordinate parameter goes on
                            ; stack
      FLD REC_X             ; X-coordinate parameter goes on
                            ; stack
      CALL mqerAT2          ; angle is computed
      FSTP POLAR_THETA      ;              and stored

      FLD REC_Y             ; Y onto stack
      FMUL ST,ST            ; Y is squared
      FLD REC_X             ; X onto stack
      FMUL ST,ST            ; X is squared
      FADDP ST(1),ST        ; sum of squares
      FSQRT                 ; radius is on stack
      FSTP POLAR_R          ; radius is stored and stack is
                            ; popped

      ; Now POLAR_THETA = PI/4 and
      ; POLAR_R = 3 * Squareroot(2).
```

ATN — mqerATN x = arc tangent(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerATN returns the number, in radians, whose tangent is equal to the input value. Numbers are chosen between $-\pi/2$ and $\pi/2$. All zeroes, pseodo-zeroes, and denormals return the input value. Also, unnormals less than 2^{-63} return the input value.

Infinite inputs are valid, whether the 80287 is in affine (signed infinity) or projective (unsigned infinity) mode. The input −INFINITY returns the value $-\pi 2$; the input +INFINITY returns the value $+\pi 2$.

Output

The answer replaces the input on the 80287 stack.

Errors

An input NaN gives an I error. Also, an unnormal not less than 2^{-63} gives an I error. If I is masked, the answer returned is the input for NaNs; the answer is the value INDEFINITE for illegal unnormals. If I is unmasked, the trap handler is called with the input still on the 80287 stack and the value 176 hex in the 80287 instruction selector.

Example of PL/M-286 Use

```
mqerATN: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerATN;

DECLARE OPPOSITE_SIDE REAL;
DECLARE ADJACENT_SIDE REAL;      /* Shorter sides of a right
                                    triangle */
DECLARE THETA_RADIANS REAL;

OPPOSITE_SIDE = 1.; ADJACENT_SIDE = 1.;   /* Test values */

/* The following line computes the angle of a right
   triangle, given the lengths of the two shorter
   sides. */

THETA_RADIANS = mqerATN(OPPOSITE_SIDE/*ADJACENT_SIDE);

/* THETA_RADIANS now equals the value of the angle:
   PI/4. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerATN: FAR
```

```
OPPOSITE_SIDE   DQ   5.0         ; short side, opposite angle
                                 ; computed
ADJACENT_SIDE   DQ   5.0         ; short side, next to angle
                                 ; computed
                     ; The above initial values are test
                     ; values
THETA_RADIANS   DQ   ?

    ; The following lines compute an angle of a right
    ; triangle, just as in the above PL/M example, except
    ; with LONG_REAL variables.

       FLD OPPOSITE_SIDE         ; (x) := OPPOSITE_SIDE
       FDIV ADJACENT_SIDE        ; (x) = OPPOSITE_SIDE/
                                 ; ADJACENT_SIDE
       CALL mqerATN              ; angle is now in (x)
       FSTP THETA_RADIANS        ; radians are saved, 80287
                                 ; stack popped

    ; With the test inputs, THETA_RADIANS = PI/4.
    ; It is a 45-45-90 triangle.
```

COS — mqerCOS x = cosine(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerCOS returns the trigonometric cosine of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the value 1. Also, unnormals whose value is less than 2^{-63} return the value 1.

Output

The answer replaces the input on the 80287 stack.

Errors

An I error is given for input infinities and NaNs. An I error is also given for unnormals that do not represent values less than 2^{-63}.

If I is unmasked, the trap handler is called with the input still on the stack, and the 80287 opcode register is set to 172 hex. If I is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

Example of PL/M-286 Use

```
mqerCOS: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerCOS;

DECLARE (POLAR_R, POLAR_THETA) REAL;
DECLARE REC_X REAL;
DECLARE PI LITERALLY '3.14159265358979';
DECLARE DEG_TO_RAD LITERALLY 'PI/180.';

POLAR_R = 2.; POLAR_THETA = 30.;    /* Test values */

/* The following line computes the X-coordinate of a
   polar-to-rectangular conversion. The input angle is in
   degrees, so it must be converted to radians. */

REC_X = POLAR_R * mqerCOS(POLAR_THETA * DEG_TO_RAD);

/* Now in the test case, REC_X = the square root of 3. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerCOS: FAR

POLAR_THETA    DQ   30.0
POLAR_R        DQ   2.0
                    ; the above initializations are test
                    ; values.
```

```
REC_X           DQ   ?
DEG_TO_RAD      DT   3FF98EFA351294E9C8AER    ; the constant
                                              ; PI/180.

    ; The following lines compute the X-coordinate of a
    ; polar-to-rectangular conversion, as in the PL/M
    ; example above; except that the variables are
    ; LONG_REAL.

    FLD POLAR_THETA       ; degrees angle onto 80287 stack
    FLD DEG_TO_RAD
    FMUL                  ; converted to radians
    CALL mqerCOS          ; cosine is taken
    FMUL POLAR_R          ; answer scaled to correct radius
    FSTP REC_X            ; X-coordinate stored and stack is
                          ; popped

    ; With the test case, REC_X is now the square root of
    ; 3.
```

CSH — mqerCSH x = hyperbolic cosine(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerCSH returns the hyperbolic cosine of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the value 1. Also, unnormals whose value is less than 2^{-63} return the value 1.

Either infinity returns the value +INFINITY with no error.

Output

The answer replaces the input on the 80287 stack.

Errors

An I error is given for input NaNs. An I error is also given for unnormals that do not represent values less than 2^{-63}.

If I is unmasked, the trap handler is called with the input still on the stack. If I is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for invalid unnormals.

mqerCSH gives an O overflow error if the input is greater than about 11355, or less than about −11355. When O is masked, the value +INFINITY is returned. When O is unmasked, the trap handler is called with the input still on the 80287 stack.

When the trap handler is called, mqerSNH first sets the 80287 instruction selector to 16F hex.

Example of PL/M-286 Use

```
mqerCSH: PROCEDURE (THETA) REAL EXTERNAL;
   DECLARE THETA REAL;
END mqerCSH;

DECLARE (INPUT_VALUE, OUTPUT_VALUE) REAL;

INPUT_VALUE = 1.3; /* Test value */

OUTPUT_VALUE = mqerCSH(INPUT_VALUE);

/* For the test case, OUTPUT_VALUE now is approximately
    1.97091 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerCSH: FAR
```

```
INPUT_VALUE      DQ  1.56    ; test value
OUTPUT_VALUE     DQ  ?

   FLD INPUT_VALUE       ; input goes onto 80287 stack
   CALL mqerCSH          ; hyperbolic cosine is taken
   FSTP OUTPUT_VALUE     ; answer is stored in 80286 memory,
                         ; 80287 is popped

; For the test value, OUTPUT_VALUE now is about 2.48448
```

DIM — mqerDIM $x = \max(y - x, +0)$

Input Parameters

(x) is the top number on the 80287 stack; (y) is the next number on the 80287 stack.

Function

mqerDIM first compares (y) and (x); if (x) is greater than (y), the answer +0 is returned. If (y) is greater than (x), the positive value (y − x) is returned.

When the 80287 chip is in affine (signed infinity) mode, and when both inputs are not the same, infinite inputs are allowed. The results are as expected from the above comparison and subtraction. That is, if (x) is +INFINITY or (y) is −INFINITY, the answer is +0; if (x) is −INFINITY or (y) is +INFINITY, the answer is +INFINITY.

mqerDIM uses the 80287 precision and rounding modes that are in effect when mqerDIM is called. Thus, the results can vary, depending on the settings of these bits in the 80287 control word.

Output

The 80287 stack pops once, with the answer replacing the two inputs.

Errors

The first error detected is the D error, which is for unnormal or denormal inputs. If the 80287 is in warning mode (D is masked), the calculation continues, but the D error bit remains set. If the 80287 is in normalizing mode (D is unmasked), the trap handler is called. The trap handler is called directly from the interior of mqerDIM; the 80287 instruction selector contains the code for the instruction that found the denormal: either FCOM or FSUB. The (y) and (x) inputs are left on the 80287 stack. Most trap handlers replace the denormal input(s) with normalized numbers, reexecute the FCOM or FSUB instruction, and continue through mqerDIM. The trap handler provided by EH287.LIB (described in Chapter 5) does these things and replaces denormals with zero.

mqerDIM next checks for NaN inputs. If either input is a NaN, an I error is given. If I is masked, the answer returned is the input NaN (the larger NaN if both inputs are NaNs).

An I error is also given if the 80287 chip is in projective (unsigned infinity) mode and either input is infinity, or if the 80287 chip is in affine (signed infinity) mode and both inputs are the same infinity. In these cases, a masked I yields the value INDEFINITE for an answer.

For all I errors, when I is unmasked, the trap handler is called with the inputs still on the 80287 stack, and the 80287 instruction selector is set to 265 hex.

In the case (y) > (x), when the answer is calculated by the subtraction (y − x), the subtraction can cause either an O overflow error or a U underflow error. If the error is masked, the answer returned is +INFINITY for overflow. For underflow, a gradual-underflow denormal is returned if possible; 0 is returned if not possible. If the error is unmasked, the trap handler is called, with the value 165 hex in the 80287 instruction selector. The inputs are not on the 80287 stack. Instead, the correct answer

appears on the 80287 stack, with the out-of-range exponent given "wrapped around." For underflow, the correct exponent is the given exponent minus the decimal value 24576; for overflow, it is the exponent plus 24576.

Example of PL/M-286 Use

```
mqerDIM: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerDIM;

DECLARE (COSTS, RECEIPTS) REAL;
DECLARE (PROFIT, LOSS) REAL;

COSTS = 4700.00; RECEIPTS = 5300.00; /* Test values */

/* The following lines return the profit or loss, given
   the costs and receipts. The positive difference goes
   into PROFIT or LOSS as appropriate, and the other value
   is set to zero. */

PROFIT = mqerDIM(RECEIPTS,COSTS);
LOSS = mqerDIM(COSTS,RECEIPTS);

/* For the test case, PROFIT is now 600.00, and LOSS
   is 0. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerDIM: FAR

COSTS        DQ   800.00
RECEIPTS     DQ   650.00
                  ; the above initializations are test cases
PROFIT       DQ   ?
LOSS         DQ   ?

   ; The following lines compute profit and loss, just as
   ; in the PL/M example.

   FLD RECEIPTS       ; first parameter to mqerDIM
   FLD COSTS          ; second parameter to mqerDIM
   CALL mqerDIM       ; positive difference is computed
   FSTP PROFIT        ; answer is stored and stack is popped
   FLD COSTS          ; now perform function the other way
   FLD RECEIPTS
   CALL mqerDIM
   FSTP LOSS          ; other answer goes to LOSS, and stack
                      ; is again popped

   ; For this test case, LOSS = 150.00 and PROFIT is 0.
```

EXP — mqerEXP $x = e^x$

Input Parameters

(x) is the top number on the 80287 stack.

Function

mqerEXP raises the constant e (2.718281828459+) to the power of the input number. This is a standard function because it is used to derive other functions, notably the hyperbolics. Also, on many computers it is slightly easier to compute than exponentials based on other constants. The exponential is valid for both positive and negative inputs.

All input zeroes, pseudo zeroes, denormals, and unnormals less than 2^{-63} yield an answer of 1.

If the 80287 is in affine (signed infinity) mode, infinite inputs are valid. +INFINITY returns itself as the answer; −INFINITY returns 0.

Output

The answer replaces the input on the 80287 stack.

Errors

All input NaNs and input unnormals greater than 2^{-63} cause an I error. Also, if the 80287 is in projective (unsigned infinity) mode, both infinite inputs give an I error. If I is masked, the value returned is the input if a NaN; the value INDEFINITE otherwise. If I is unmasked, the trap handler is called with the input still on the 80287 stack.

Because the largest number that can be stored in the 80287's temporary real format is approximately e^{11357}, an input greater than about 11357 causes an O overflow error. When O is masked, the value +INFINITY is returned. Likewise, an input less than about −11355 causes the U underflow error. If U is masked, the result is a gradual-underflow denormal, if possible; zero otherwise. For unmasked O or U errors, the trap handler is called with the input still on the 80287 stack.

When the trap handler is called, mqerEXP first places the value 16B hex into the 80287 instruction register.

Example of PL/M-286 Use

```
mqerEXP: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerEXP;

DECLARE Y_VALUE REAL;
DECLARE X_VALUE REAL;
DECLARE NORM_CONSTANT LITERALLY '0.3989422803';
/* 1 / (SQRT(2*PI)) */

X_VALUE = -2.0;    /* Test value */
```

```
/* The following line gives the value for the graph of the
   normal distribution function, for mean 0 and variance
   1. By plotting Y_VALUE against various different input
   X-values, one obtains the familiar ``bell-shaped
   curve''. */

Y_VALUE = NORM_CONSTANT
            * mqerEXP( - X_VALUE * X_VALUE / 2. );

/* For the test input, Y_VALUE is now approximately
   0.5399100 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerEXP: FAR

MINUS_2          DQ   -2.   ; constant used in calculation
                            ; below
X_VALUE          DQ   0.4   ; initialization is a test value
Y_VALUE          DQ   ?
NORM_CONSTANT    DT   3FFDCC42299EA1B28468R   ; constant
                                              ; 1/sqrt(2*PI)

   ; The following lines compute the normal distribution
   ; just as in the PL/M example, except with LONG_REAL
   ; numbers.

   FLD X_VALUE            ; load input onto 80287 stack
   FMUL ST,ST             ; input is squared
   FIDIV MINUS_2          ; negate and divide by 2
   CALL mqerEXP           ; exponentiate
   FLD NORM_CONSTANT
   FMUL                   ; divide by the square root of
                          ; 2 * PI
   FSTP Y_VALUE           ; store the result and pop the
                          ; stack

   ; For the test case Y_VALUE is now about 0.3399562
```

IA2 — mqerIA2 AX = roundaway(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, x is rounded to the nearest integer. If two integers are equally near (i.e., the fractional part of x is .5), the integer farthest from zero is selected. The answer is given in 16-bit two's complement form. For example, 3.1 returns hex 0003; 10.5 returns hex 000B; −2.5 returns hex FFFD, which is −3 in two's complement form.

Output

The input is popped from the 80287 stack and the answer is left in the AX register.

Errors

The only numbers that will fit the 16-bit destination are those between, but not including, −32768.5 and +32767.5. Numbers not in this range, including infinities and NaNs, cause an I error. If I is masked, the "indefinite integer" value 8000 hex is returned. If I is unmasked, the trap handler is called with the input still on the 80287 stack, and the 80287 instruction selector is set to 17E hex.

Example of PL/M-286 Use

```
mqerIA2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIA2;

DECLARE REAL_VAR REAL;
DECLARE INTEGER_VAR INTEGER;

REAL_VAR = 4.5;    /* Test value */

INTEGER_VAR = mqerIA2(REAL_VAR);

/* Now in the test case, INTEGER_VAR = 0005 hex. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIA2: FAR

INTEGER_VAR    DW   ?
REAL_VAR       DQ   -8.5   ; Initialization is a test value

   FLD REAL_VAR            ; load the parameter
   CALL mqerIA2            ; round to nearest integer
   MOV INTEGER_VAR,AX      ; store the answer

   ; For the test case, INTEGER_VAR now equals -9, which
   ; is FFF7 hex.
```

IA4 — mqerIA4 DXAX = roundaway(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, x is rounded to the nearest integer. If two integers are equally near (i.e., the fractional part of x is .5), the integer farthest from zero is selected. The answer is given in 32-bit two's complement form. For example, 3.1 returns hex 00000003; 10.5 returns hex 0000000B; −2.5 returns hex FFFFFFFD, which is −3 in two's complement form.

Output

The input is popped from the 80287 stack, and the answer is left in the DX and AX registers, with DH the highest byte and AL the lowest byte.

Errors

The only numbers that will fit the 32-bit destination are those between, but not including, −2147483648.5 and +2147483647.5. Numbers not in this range, including infinities and NaNs, cause an I error. If I is masked, the "indefinite integer" value 80000000 hex is returned. If I is unmasked, the trap handler is called with the input still on the 80287 stack, and the 80287 instruction selector is set to 168 hex.

Example of PL/M-286 Use

Because PL/M-286 does not support a 32-bit integer data type, mqerIA4 cannot be used by PL/M programs. You should use either mqerIA2 or mqerIAX.

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIA4: FAR

COUNT          DD   ?
REAL_COUNT     DQ   100000.5    ; initialized to a test value

; The following code assumes that the above variables are
; in the DS segment.

   FLD REAL_COUNT              ; load the parameter onto the
                               ; 80287 stack
   CALL mqerIA4                ; convert the number to 32 bits.
   MOV BX,OFFSET(COUNT)        ; point to the destination
   MOV [BX],AX                 ; move lower 16 bits of answer
   MOV [BX+2],DX               ; move upper 16 bits of answer

   ; With the test input, COUNT is now 100001, which is
   ; 000186A1 hex.
```

IAX — mqerIAX x = roundaway(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, x is rounded to the nearest integer. If two integers are equally near (i.e., the fractional part of x is .5), the integer farthest from zero is selected. For example, 3.3 returns 3; 4.5 returns 5; −6.5 returns −7.

Infinite values are returned unchanged, with no error.

Output

The answer replaces the input on the 80287 stack.

Errors

The I error is set if the input is a NaN. The NaN is left unchanged. If the I error is unmasked, the trap handler is called with the 80287 instruction selector set to 167 hex.

Example of PL/M-286 Use

```
mqerIAX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerIAX;

DECLARE (HOURS, MINUTES) REAL;

HOURS = 2.71;   /* Test value */

/* The following statement converts HOURS into an integer
   number of MINUTES */

MINUTES = mqerIAX(HOURS*60.);

/* Now MINUTES = 163. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIAX: FAR

HOURS        DQ   2.71    ; initialized to test value
MINUTES      DQ   ?
SIXTY        DD   60.0

; The following lines convert HOURS into an integer number
; of MINUTES. It does the same thing as the above PL/M
; example, only with LONG_REAL numbers.
```

```
        FLD HOURS          ; put HOURS onto 80287 stack
        FMUL SIXTY         ; convert to a real number of MINUTES
        CALL mqerIAX       ; round to the nearest integer
        FSTP MINUTES       ; store the integer in LONG_REAL format

    ; With the test case, MINUTES is now 163.0
```

IC2 — mqerIC2 AX = chop(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, if x is an integer, it is returned unchanged. If x is not an integer, the fractional part of x is chopped. Thus, the nearest integer in the direction of 0 is returned. The answer is given in 16-bit two's complement form. For example, 4 returns hex 0004; 11.7 returns hex 000B; −6.9 returns hex FFFA, which is −6 in two's complement form.

Output

The input is popped from the 80287 stack, and the answer is left in the AX register.

Errors

The only numbers that will fit the 16-bit destination are those between, but not including, −32769 and +32768. Numbers not in this range, including infinities and NaNs, cause an I error. If I is masked, the "indefinite integer" value 8000 hex is returned. If I is unmasked, the trap handler is called with the input still on the 80287 stack.

Note that "indefinite integer" can also represent the legal output value −32768. The two outputs can be distinguished only by the setting of the I exception bit of the 80287.

If a truncation does take place, the P error is set. If P is masked, the answer is returned as usual, because this is usually considered no error. If P is unmasked, the trap handler is called. Because the output in the AX register is likely to be lost before the trap handler can use it, it is best not to unmask the P exception.

If either trap handler is called, the 80287 instruction selector is first set to 17E hex.

Example of PL/M-286 Use

```
mqerIC2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIC2;

DECLARE CONTROL_SETTING INTEGER;
DECLARE REAL_INPUT REAL;

REAL_INPUT = 37.885;    /* Test value */

/* The following line translates REAL_INPUT, which could
   have been calculated using floating-point arithmetic,
   into an INTEGER value CONTROL_SETTING, which might be
   output as up to 16 logic lines to a physical device. */

CONTROL_SETTING = mqerIC2(REAL_INPUT);

/* For the test input, CONTROL_SETTING is now 37; which is
0025 hex. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIC2: FAR

REAL_INPUT          DQ   65.7    ; initialized to a test
                                 ; value
CONTROL_SETTING     DW   0

; The following lines convert the REAL_INPUT into the
; 16-bit integer value CONTROL_SETTING, just as in the
; PL/M example above, except that REAL_INPUT is a
; LONG_REAL number.

   FLD REAL_INPUT               ; load the input onto the 80287
                                ; stack
   CALL mqerIC2                 ; chop to the integer part
   MOV CONTROL_SETTING,AX       ; store the 16-bit answer

; For the input test value, CONTROL_SETTING is now 65, or
; 41 hex.
```

IC4 — mqerIC4 DXAX = chop(x)

Input Parameter

x is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, if x is an integer, it is returned unchanged. If x is not an integer, the fractional part of x is chopped. Thus, the nearest integer in the direction of 0 is returned. The answer is given in 32-bit two's complement form. For example, 4 returns hex 00000004; 11.7 returns hex 0000000B; −6.9 returns hex FFFFFFFA, which is −6 in two's complement form.

Output

The input is popped from the 80287 stack, and the answer is left in the DX and AX registers, with DH the highest byte and AL the lowest byte.

Errors

The only numbers that will fit the 32-bit destination are those between, but not including, −2147483649 and +2147483648. Numbers not in this range, including infinities and NaNs, cause an I error. If I is masked, the "indefinite integer" value 80000000 hex is returned. If I is unmasked, the trap handler is called with the input still on the 80287 stack.

Note that "indefinite integer" can also represent the legal output value −2147483648. The two outputs can be distinguished only by the setting of the I exception bit of the 80287.

If a truncation does take place, the P error is set. If P is masked, the answer is returned, as usual, because this is usually considered no error. If P is unmasked, the trap handler is called. Because the output in the DXAX registers is likely to be lost before the trap handler can use it, it is best not to unmask the P exception.

If either trap handler is called, the 80287 instruction selector is first set to 179 hex.

Example of PL/M-286 Use

Because PL/M-286 supports no 32-bit integer data type, mqerIC4 cannot be used by PL/M programs. You should use either mqerIC2 or mqerICX.

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIC4: FAR

POPULATION          DQ   5306279.0     ; total number of voters
SUPPORT_SHARE       DQ   .39           ; proportion of
                                       ; population supporting
                                       ; us above numbers are
                                       ; initialized to test
                                       ; values
```

```
SUPPORT_VOTES       DD   ?

; The following lines calculate the number of voters
; supporting an issue, given the total population, and the
; fractional share of the population which supports the
; issue. This is simply the share multiplied by the
; total, then chopped to a 32-bit integer SUPPORT_VOTES.

   FLD POPULATION                        ; load total onto
                                         ; 80287 stack
   FMUL SUPPORT_SHARE                    ; multiply by share
   CALL mqerIC4                          ; chop to 32 bits
   MOV WORD PTR SUPPORT_VOTES, AX        ; store bottom 16
                                         ; bits
   MOV WORD PTR (SUPPORT_VOTES+2), DX ; store top 16 bits

; With the test inputs, SUPPORT_VOTES is now 2069448,
; which is 001F93C8 hex.
```

ICX — mqerICX x = chop(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, if x is an integer, it is returned unchanged. If x is not an integer, the fractional part of x is chopped. Thus, the nearest integer in the direction of 0 is returned. For example, 4 returns 4; −3.9 returns −3; 1.7 returns 1.

Infinite values are returned unchanged, with no error.

Output

The answer replaces the input on the 80287 stack.

Errors

If a truncation does take place, the P error is set. The correct answer is on the 80287 stack. If P is unmasked (this is rarely done), the trap handler is then called.

The I error is set if the input is a NaN. The NaN is left unchanged; if the I error is unmasked, the trap handler is called.

If either trap handler is called, the 80287 instruction selector is first set to 166 hex.

Example of PL/M-286 Use

```
mqerICX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerICX;

DECLARE PRICE_DOLLARS REAL;
PRICE_DOLLARS = 37.596;    /* Test value */

/* The following statement chops PRICE_DOLLARS to an even
   number of pennies. First, PRICE_DOLLARS is multiplied
   by 100 to get a number of pennies; second, the pennies
   are chopped by mqerICX; third, the answer is divided by
   100 to convert back into dollars. */

PRICE_DOLLARS = ( mqerICX(PRICE_DOLLARS * 100.) / 100. ) ;

/* Now PRICE_DOLLARS = 37.59 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerICX: FAR

PRICE_DOLLARS     DQ   37.596     ; initialized to a test
                                  ; value
ONE_HUNDRED       DD   100.00     ; constant which is used
                                  ; twice below
```

```
; The following lines chop PRICE_DOLLARS to an even number
; of pennies, just as in the PL/M example above, except
; that PRICE_DOLLARS is here a LONG_REAL variable.

   FLD PRICE_DOLLARS        ; amount is loaded onto 80287
                            ; stack
   FMUL ONE_HUNDRED         ; amount is converted to pennies
   CALL mqerICX             ; pennies are chopped
   FDIV ONE_HUNDRED         ; amount is converted back to
                            ; dollars
   FSTP PRICE_DOLLARS       ; converted number is stored,
                            ; 80287 stack is popped

; With the above test input, PRICE_DOLLARS is now 37.59
```

IE2 — mqerIE2 AX = roundeven(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, x is rounded to the nearest integer. If two integers are equally near (i.e., the fractional part of x is .5), the even integer is selected. The answer is given in 16-bit two's complement form. For example, 3.1 returns hex 0003; 10.5 returns hex 000A; −2.5 returns hex FFFE, which is −2 in two's complement form.

Output

The input is popped from the 80287 stack and the answer is left in the AX register.

Errors

The only numbers that will fit the 16-bit destination are those greater than or equal to −32768.5 and less than +32767.5. Numbers not in this range, including infinities and NaNs, cause an I error. If I is masked, the "indefinite integer" value 8000 hex is returned. If I is unmasked, the trap handler is called with the input still on the 80287 stack.

If a rounding does take place, the P error is set. If P is masked, the answer is returned, as usual, because this is usually considered no error. If P is unmasked, the trap handler is called. Because the output in the AX register is likely to be lost before the trap handler can use it, it is best not to unmask the P exception.

When the trap handler is called, the 80287 instruction selector is first set to 180 hex.

Example of PL/M-286 Use

```
mqerIE2: PROCEDURE (X) INTEGER EXTERNAL;
  DECLARE X REAL;
END mqerIE2;

DECLARE REAL_VAR REAL;
DECLARE INTEGER_VAR INTEGER;

REAL_VAR = 4.5;    /* Test value */

INTEGER_VAR = mqerIE2(REAL_VAR);

/* Now in the test case, INTEGER_VAR = 0004 hex. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIE2: FAR

INTEGER_VAR    DW   ?
REAL_VAR       DQ   -8.5    ; Initialization is a test value
```

```
FLD REAL_VAR             ; load the parameter
CALL mqerIE2             ; round to nearest integer
MOV INTEGER_VAR,AX       ; store the answer

; For the test case, INTEGER_VAR now equals -8, which
; is FFF8 hex.
```

IE4 — mqerIE4 DXAX = roundeven(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, x is rounded to the nearest integer. If two integers are equally near (i.e., the fractional part of x is .5), the even integer is selected. The answer is given in 32-bit two's complement form. For example, 3.1 returns hex 00000003; 10.5 returns hex 0000000A; −2.5 returns hex FFFFFFFE, which is −2 in two's complement form.

Output

The input is popped from the 80287 stack, and the answer is left in the DX and AX registers, with DH the highest byte and AL the lowest byte.

Errors

The only numbers that will fit the 32-bit destination are those greater than or equal to −2147483648.5 and less than +2147483647.5. Numbers not in this range, including infinities and NaNs, cause an I error. If I is masked, the "indefinite integer" value 80000000 hex is returned. If I is unmasked, the trap handler is called with the input still on the 80287 stack.

If a rounding does take place, the P error is set. If P is masked, the answer is returned, as usual, because this is usually considered no error. If P is unmasked, the trap handler is called. Because the output in the DXAX registers is likely to be lost before the trap handler can use it, it is best not to unmask the P exception.

When the trap handler is called, the 80287 instruction selector is first set to 17B hex.

Example of PL/M-286 Use

Because PL/M-286 supports no 32-bit integer data type, mqerIE4 cannot be used by PL/M programs. You should use either mqerIE2 or mqerIEX.

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIE4: FAR

COUNT          DD   ?
REAL_COUNT     DQ   100000.5    ; initialized to a test value

; The following code assumes that the above variables are
; in the DS segment.

   FLD REAL_COUNT                  ; load the parameter onto
                                   ; the 80287 stack
   CALL mqerIE4                    ; convert the number to 32
                                   ; bits.
   MOV WORD PTR COUNT, AX          ; move lower 16 bits of
```

```
                                  ; answer
MOV WORD PTR (COUNT+2), DX        ; move upper 16 bits of
                                  ; answer

; With the test input, COUNT is now 100000, which is
; 000186A0 hex.
```

IEX — mqerIEX x = roundeven(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

If x is not normal, it is first normalized. Then, x is rounded to the nearest integer. If two integers are equally near (i.e., the fractional part of x is .5), the even integer is selected. For example, 3.3 returns 3; 4.5 returns 4; −6.5 returns −6.

Infinite values are returned unchanged, with no error.

Output

The answer replaces the input on the 80287 stack.

Errors

If a rounding does take place, the P error is set. The correct answer is on the 80287 stack. If P is unmasked (this is rarely done), the trap handler is then called.

The I error is set if the input is a NaN. The NaN is left unchanged; if the I error is unmasked, the trap handler is called.

If either trap handler is called, the 80287 instruction selector is first set to 178 hex.

Example of PL/M-286 Use

```
mqerIEX: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerIEX;

DECLARE UNITS REAL;
DECLARE THOUSANDS REAL;

UNITS = 4500.00;   /* Test value */

/* The following line computes an integer number of
   thousands, given an input number of units. */

THOUSANDS = mqerIEX(UNITS/1000.);

/* With the test input value, THOUSANDS now equals 4.00 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerIEX: FAR

THOUSANDS       DQ   ?
UNITS           DQ   5890.14      ; initialization is a test
                                  ; value
ONE_GRAND       DD   1000.00      ; constant for the division
                                  ; below
```

```
; The following lines compute an integer number of
; thousands, just as in the PL/M example above, except the
; LONG_REAL variables are used.

   FLD UNITS             ; load the input onto the 80287 stack
   FDIV ONE_GRAND        ; convert to thousands
   CALL mqerIEX          ; round to an integer value
   FSTP THOUSANDS        ; store it into 80286 memory and pop
                         ; the 80287 stack

   ; With the test value, THOUSANDS is now 6.00
```

LGD — mqerLGD x = common log(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerLGD returns the number L, with 10^L equalling the input number. For positive inputs, this is a well-defined number. For example, an input 10 gives 1; 100 gives 2; .001 gives −3. Inputs between 10 and 100 give outputs between 1 and 2.

If the 80287 processor is in affine (signed infinity) mode, the input +INFINITY is also valid, returning itself as an answer.

Output

The answer replaces the input on the 80287 stack.

Errors

Negative numbers (including −INFINITY), all NaNs, and all unnormals give an I error. Also, when the 80287 is in projective (unsigned infinity) mode, both values of INFINITY give an I error. If I is masked, the result is the input for NaNs; the result is the value INDEFINITE for other invalid inputs. If I is unmasked, the trap handler is called wth the input number still on the 80287 stack.

Zero input, of either sign, gives a Z error. If Z is masked, the result is −INFINITY. If Z is unmasked, the trap handler is called with the input number still on the 80287 stack.

If the input is a denormal, mqerLGD tests the 80287 D exception bit to see if the 80287 is in normalizing mode. If in normalizing mode (D unmasked), the input is treated as a zero, and a Z error is issued. If not in normalizing mode (D masked), the input is treated as an unnormal, and an I error is issued. Note that even though the D masking bit is tested, the D error is never issued by mqerLGD, and the D trap handler is never called during mqerLGD.

Whenever the trap handler is called, the number 16D hex is first placed into the 80287 instruction selector.

Example of PL/M-286 Use

```
mqerLGD: PROCEDURE (X) REAL EXTERNAL;
   DECLARE X REAL;
END mqerLGD;

DECLARE QUANTITY REAL;
DECLARE TENS_POWER REAL;

QUANTITY = 1900.00;     /* Test value */

TENS_POWER = mqerLGD(QUANTITY);

/* Since the test value QUANTITY is between 10 ** 3 and
   10 ** 4, the answer TEST_POWER is between 3 and 4. It
   is about 3.27875. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerLGD: FAR

QUANTITY        DQ   .0001     ; initialized to a test value
TENS_POWER      DQ   ?

; The following code implements the above PL/M call in
; assembly language, with LONG_REAL variables.

   FLD QUANTITY               ; load input onto 80287 stack
   CALL mqerLGD               ; take the base 10 logarithm
   FSTP TENS_POWER            ; store the answer and pop the
                              ; 80287 stack

; Since the test input was 10 ** -4, the output should be
; -4.00. Due to accumulated rounding errors, it may not
; be the exact integer.
```

LGE — mqerLGE x = natural log(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerLGE returns the number L, with e^L equalling the input number. The constant e is 2.718281828459+. This logarithm is called "natural" because it occurs in calculus as the inverse derivative of 1/X; on many computers it is slightly easier to compute than other logarithms. For positive inputs, the logarithm is a well-defined number. If the 80287 processor is in affine (signed infinity) mode, the input +INFINITY is also valid, returning itself as an answer.

Output

The answer replaces the input on the 80287 stack.

Errors

Negative numbers (including −INFINITY), all NaNs, and all unnormals give an I error. Also, when the 80287 is in projective (unsigned infinity) mode, both values of INFINITY give an I error. If I is masked, the result is the input for NaNs; the result is the value INDEFINITE for other invalid inputs. If I is unmasked, the trap handler is called with the input number still on the 80287 stack.

Zero input, of either sign, gives a Z error. If Z is masked, the result is −INFINITY. If Z is unmasked, the trap handler is called with the input number still on the 80287 stack.

If the input is a denormal, mqerLGE tests the 80287 D exception bit to see if the 80287 is in normalizing mode. If in normalizing mode (D unmasked), the input is treated as a zero, and a Z error is issued. If not in normalizing mode (D masked), the input is treated as an unnormal, and an I error is issued. Note that even though the D masking bit is tested, the D error is never issued by mqerLGE, and the D trap handler is never called during mqerLGE.

Whenever the trap handler is called, the number 16C hex is first placed into the 80287 instruction selector.

Example of PL/M-286 Use

```
mqerLGE: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END mqerLGE;

FSQRT287: PROCEDURE (X) REAL EXTERNAL;
  DECLARE X REAL;
END FSQRT287;

/* The above procedure is simply a PUBLIC ASM286 procedure
   which consists of the FSQRT instruction, followed by a
   RET instruction. */
```

```
DECLARE X REAL;
DECLARE THETA REAL;

X = 3.0;    /* Test value */

/* The following code calculates the inverse hyperbolic
   sine of X. That is, THETA is set to the number whose
   hyperbolic sine is X. */

THETA = mqerLGE( X + FSQRT287(X X + 1.));

/* For the test input, THETA now equals about 1.81845 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerLGE: FAR

X           DQ   -3.0      ; initialized to a test value
THETA       DQ   ?
ONE         DD   1.00      ; constant used below

; The following code calculates the inverse hyperbolic
; sine, just as in the PL/M example above, except with
; LONG_REAL inputs.

   FLD X                  ; input parameter onto 80287 stack
   FMUL ST,ST             ; X squared
   FADD ONE               ; X squared + 1
   FSQRT
   FADD X                 ; X + SQRT(X squared + 1)
   CALL mqerLGE           ; take the natural logarithm -- this
                          ; is the answer
   FSTP THETA             ; store it and pop the 80287 stack

; With the test input THETA is now about -1.81845
```

MAX — mqerMAX x = max(x,y)

Input Parameters

(x) is the top number on the 80287 stack; (y) is the next number on the 80287 stack.

Function

mqerMAX returns the greater of the numbers (x) and (y). That is, if the 80287 FCOM instruction indicates that (x) > (y), then (x) is returned. If (x) < (y), then, (y) is returned. When (x) and (y) test as equal—even though they may be given in different formats (for example, +0 and −0),—(x) is returned.

If the 80287 chip is in affine (signed infinity) mode, either or both inputs can be infinite. If the 80287 is in projective (unsigned infinity) mode, and if either input is infinite, both must be infinite. In that case, the values test as equal and (x) is the answer.

Output

The 80287 stack pops once, with the answer replacing the two inputs.

Errors

The first error detected is the D error, which is for unnormal or denormal inputs. If the 80287 is in warning mode (D is masked), the calculation continues, but the D error bit remains set. If the 80287 is in normalizing mode (D is unmasked), the D trap handler is called. The trap handler is called directly from the interior of mqerMAX; the 80287 opcode register contains the code for the FCOM instruction that caused the D error. The (y) and (x) inputs are left on the 80287 stack. Most trap handlers replace the denormal input(s) with normalized numbers, reexecute the FCOM or FSUB instruction, and continue through mqerMAX. The trap handler provided by EH287.LIB (described in Chapter 5) does these things and replaces denormals with zero.

mqerMAX next checks for NaN inputs. If either input is a NaN, an I error is given. If I is masked, the answer returned is the input NaN (the larger NaN if both inputs are NaNs).

An I error is also given if the 80287 chip is in projective (unsigned infinity) mode, and one of the inputs is infinite. In this case, a masked I yields the value INDEFINITE for an answer.

For all I errors when I is unmasked, the trap handler is called with the inputs still on the 80287 stack, and the 80287 instruction selector set to 265 hex.

Example of PL/M-286 Use

```
mqerMAX: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMAX;

DECLARE POPULATIONS(10) REAL;
DECLARE LARGEST REAL;
DECLARE N BYTE;
```

```
/* The following code sets LARGEST to the greatest of the
   ten values in the array POPULATIONS. */

LARGEST = POPULATIONS(0);
DO N = 1 TO 9;
  LARGEST = mqerMAX(LARGEST,POPULATIONS(N));
END;
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerMAX: FAR

VAR1          DQ   -5.3
VAR2          DQ   -7.9
                    ; the above initializations are test
                    ; values
LARGEST       DQ   ?

; The following code sets LARGEST to the maximum of VAR1
; and VAR2.

     FLD VAR1              ; load the first parameter onto
                           ; the 80287 stack
     FLD VAR2              ; load the second parameter
     CALL mqerMAX          ; stack now contains the maximum
     FSTP LARGEST          ; maximum is stored, and 80287
                           ; stack is popped

; With the test inputs, LARGEST is now -5.3
```

MIN — mqerMIN x = min(x,y)

Input Parameters

(x) is the top number on the 80287 stack; (y) is the next number on the 80287 stack.

Function

mqerMIN returns the lesser of the numbers (x) and (y). That is, if the 80287 FCOM instruction indicates that (x) < (y), then (x) is returned. If (x) > (y), then (y) is returned. When (x) and (y) test as equal—even though they may be given in different formats (for example, +0 and −0)—(x) is returned.

If the 80287 chip is in affine (signed infinity) mode, either or both inputs can be infinite. If the 80287 is in projective (unsigned infinity) mode, and if either input is infinite, both must be infinite. In that case, the values test as equal, and (x) is the answer.

Output

The 80287 stack pops once, with the answer replacing the two inputs.

Errors

The first error detected is the D error, which is for unnormal or denormal inputs. If the 80287 is in warning mode (D is masked), the calculation continues, but the D error bit remains set. If the 80287 is in normalizing mode (D is unmasked), the D trap handler is called. The trap handler is called directly from the interior of mqerMIN; the 80287 instruction selector contains the code for the FCOM instruction that caused the D error. The (y) and (x) inputs are left on the 80287 stack. Most trap handlers replace the denormal input(s) with normalized numbers, reexecute the FCOM or FSUB instruction, and continue through mqerMIN. The trap handler provided by EH287.LIB (described in Chapter 5) does these things and replaces denormals with zero.

mqerMIN next checks for NaN inputs. If either input is a NaN, an I error is given. If I is masked, the answer returned is the input NaN (the larger NaN if both inputs are NaNs).

An I error is also given if the 80287 chip is in projective (unsigned infinity) mode and exactly one of the inputs is infinite. In this case, a masked I yields the value INDEFINITE for an answer.

For all I errors when I is unmasked, the trap handler is called with the inputs still on the 80287 stack and the 80287 instruction selector set to 265 hex.

Example of PL/M-286 Use

```
mqerMIN: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerMIN;

DECLARE POPULATIONS(10) REAL;
DECLARE SMALLEST REAL;
DECLARE N BYTE;
```

```
/* The following code sets SMALLEST to the smallest of the
   ten values in the array POPULATIONS. */

SMALLEST = POPULATIONS(0);
DO N = 1 TO 9;
  SMALLEST = mqerMIN(SMALLEST,POPULATIONS(N));
END;
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerMIN: FAR

VAR1            DQ   -5.3
VAR2            DQ   -7.9
                        ; the above initializations are test
                        ; values
SMALLEST        DQ   ?

; The following code sets SMALLEST to the minimum of VAR1
; and VAR2.

   FLD VAR1              ; load the first parameter onto
                         ; the 80287 stack
   FLD VAR2              ; load the second parameter
   CALL mqerMIN          ; stack now contains the minimum
   FSTP SMALLEST         ; minimum is stored, and 80287
                         ; stack is popped

; With the test inputs, SMALLEST is now -7.9
```

MOD — mqerMOD x = (y mod x), same sign as (y)

Input Parameters

(x) is the top number on the 80287 stack; (y) is the next number on the 80287 stack.

Function

mqerMOD returns the answer (y−(x mqerICX(y/x)). In other words, this is the "remainder" left when (y) is divided by (x). The answer is always exact; no roundoff error occurs.

mqerMOD always returns the same values when (x) is negative as it does for the corresponding positive input (−x). (x) in the following paragraphs means the absolute value of (x).

mqerMOD is calculated by subtracting an integer multiple of (x) from the input (y) to bring it down to within (x) units of zero. The choice of which integer multiple to subtract determines the range of possible values the function can yield. For mqerMOD, the integer is mqerICX(y/x), which is the value (y/x) chopped towards zero. If (y) is postive, the answer is greater than or equal to 0, and less than (x). If (y) is negative, the answer is greater than (−x) and less than or equal to 0.

For example, suppose (x) equals either −5 or 5. Then, for negative y, mqerMOD(y,5) gives values in the range between −5 and 0. mqerMOD(−7,5) is −2; mqerMOD(−10,5) is 0; mqerMOD(−19.99,5) is −4.99. For positive y, mqerMOD(y,5) gives values in the range between 0 and 5. mqerMOD(2,5) is 2; mqerMOD(45,5) is 0; mqerMOD(44.75,5) is 4.75.

It is legal to have infinite (x) inputs. In that case, mqerMOD simply returns (y) if (y) is finite and normal. If (y) is unnormal, the normalized (y) is returned. If (y) is denormal, the result depends on the setting of the 80287's normalization mode, as determined by the D error masking flag. If in normalizing mode (D unmasked), the result is 0 with no error. If in warning mode (D masked), the result is the unchanged denormal (y), also with no error.

It is often legal to have unnormal and denormal (y) inputs. The cases with infinite (x) are discussed in the above paragraph. If (y) is unnormal, and if the normalization of (y) does not produce a denormal, then (y) is legal and the normalized input is used.

Output

The 80287 stack pops once, with the answer replacing the two inputs.

Errors

First, the inputs are checked to see if either is a NaN. If so, an I error is given. If I is masked, the input NaN is returned (if both inputs are NaNs, the larger NaN is returned).

If (x) is unnormal, denormal, or any zero value, an I error is given. Also, if (y) is infinite, an I error is given. If I is masked, the value INDEFINITE is returned.

If I is unmasked for any of the above errors, the trap handler is called, with the inputs still on the 80287 stack and the number 269 hex in the 80287 opcode register.

A U error is given when (y) is unnormal, and the normalization of (y) produces a denormal. A U error is also given if (y) is denormal. If U is masked, a 0 is returned. If U is unmasked, the trap handler is called with the value 169 hex placed in the 80287 instruction selector—but the inputs are not on the 80287 stack. Instead, the correct answer is given, with a "wrapped" exponent. To obtain the correct exponent, subtract the decimal number 24576 from the given exponent.

Example of PL/M-286 Use

```
mqerMOD: PROCEDURE (Y,X) REAL EXTERNAL;
   DECLARE (Y,X) REAL;
END mqerMOD;

DECLARE Y REAL;
DECLARE LAST_THREE_DIGITS REAL;

Y = 456789.00;   /* Test value */

/* The following line sets LAST_THREE_DIGITS to Y MOD
    1000. If Y is a positive integer, then the answer is
    the number formed by the last three decimal digits of
    Y. */

LAST_THREE_DIGITS = mqerMOD(y,1000.);

/* With the test value, LAST_THREE_DIGITS is now 789.00 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerMOD: FAR

LAST_THREE_DIGITS      DQ   ?
Y                      DQ   181137.00     ; initialization is a
                                          ; test value
ONE_GRAND              DD 1000.00         ; constant for
                                          ; calculation below

; The following code calculates Y MOD 1000, as in the PL/M
; example above, except with LONG_REAL variables

    FLD Y                  ; load first parameter onto 80287
                           ; stack
    FLD ONE_GRAND          ; load modulus 1000 onto 80287 stack
    CALL mqerMOD           ; take the modulus
    FSTP LAST_THREE_DIGITS    ; store answer and pop the
                              ; 80287 stack

; With the test value, LAST_THREE_DIGITS is 137.00
```

RMD — mqerRMD x = (y mod x), close to 0

Input Parameters

(x) is the top number on the 80287 stack; (y) is the next number on the 80287 stack.

Function

mqerRMD returns the answer (y−(x mqerIEX(y/x)). In other words, this is the "remainder" left when (y) is divided by (x). The answer is always exact; no roundoff error occurs.

mqerRMD always returns the same values when (x) is negative as it does for the corresponding positive input (−x). (x) in the following paragraphs means the absolute value of (x).

mqerRMD is calculated by subtracting an integer multiple of (x) from the input (y) to bring it down to within (x) units of zero. The choice of which integer multiple to subtract determines the range of possible values the function can yield. For mqerRMD, the integer is mqerIEX(y/x), which is the value (y/x) rounded to the nearest integer. Thus, the range of answers is from (−x/2) to (x/2).

For example, suppose (x) equals either −5 or 5. Then, the value of mqerRMD(y,5) ranges from −2.5 to 2.5. mqerRMD(−7,5) is −2; mqerRMD(−10,5) is 0; mqerRMD(−19.99,5) is +0.01; mqerRMD(2,5) is 2; mqerRMD(4,5) is −1; mqerRMD(44.75,5) is −0.25.

When the input (y) is an odd integer multiple of (x/2), the answer returned by mqerRMD can be either (x/2) or (−x/2). The number chosen is determined by the convention of mqerIEX, which rounds to the even integer in case of a tie. This results in values alternating between (−x/2) and (x/2). Inputs (y),which yield (x/2), form the series { ... −7x/2, −3x/2, x/2, 5x/2, 9x/2, ... }. Inputs (y), which yield (−x/2), form the series { ... −5x/2, −x/2, 3x/2, 7x/2, 11x/2, ... }. For example, mqerRMD(2.5,5) is 2.5; mqerRMD(7.5,5) is −2.5.

It is legal to have infinite (x) inputs. In that case, mqerRMD simply returns (y) if (y) is finite and normal. If (y) is unnormal, the normalized (y) is returned. If (y) is denormal, the result depends on the setting of the 80287's normalization mode, as determined by the D error masking flag. If in normalizing mode (D unmasked), the result is 0, with no error. If in warning mode (D masked), the result is the unchanged denormal (y), also with no error.

It is often legal to have unnormal and denormal (y) inputs. The cases with infinite (x) are discussed in the above paragraph. If (y) is unnormal, and if the normalization of (y) does not produce a denormal, (y) is legal and the normalized input is used.

Output

The 80287 stack pops once, with the answer replacing the two inputs.

Errors

First, the inputs are checked to see if either is a NaN. If so, an I error is given. If I is masked, the input NaN is returned (if both inputs are NaNs, the larger NaN is returned).

If (x) is unnormal, denormal, or any zero value, an I error is given. Also, if (y) is infinite, an I error is given. If I is masked, the value INDEFINITE is returned.

If I is unmasked for any of the above errors, the trap handler is called with the inputs still on the 80287 stack and the number 27A hex in the 80287 instruction selector.

A U error is given when (y) is unnormal and the normalization of (y) produces a denormal. A U error is also given if (y) is already denormal. If U is masked, a 0 is returned. If U is unmasked, the trap handler is called with the value 17A hex placed in the 80287 instruction selector—but the inputs are not on the 80287 stack. Instead, the correct answer is given, with a "wrapped" exponent. To obtain the correct exponent, subtract the decimal number 24576 from the given exponent.

Example of PL/M-286 Use

```
mqerRMD: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerRMD;

DECLARE TWO_PI LITERALLY '6.283185307179586476925';
     /* 2 * PI -- a full circle expressed in radians */

DECLARE THETA REAL;   /* angle to be reduced */

THETA = 6.;   /* Test value */

/* The following line reduces THETA to a principal value
   -- a value between -PI and PI.   */

THETA = mqerRMD(THETA, TWO_PI);

/* Now THETA is 6 radians, reduced to the principal value:
        about -0.2831853 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerRMD: FAR

THETA          DQ   -6.00     ; initialization is a test value

; The following code performs the same reduction of an
; angle to a principal value as the PL/M code above,
; except with a LONG_REAL variable.

   FLD THETA                  ; angle parameter onto 80287 stack
   FLDPI                      ; constant PI onto stack
   FADD ST,ST                 ; 2 * PI
   CALL mqerRMD               ; modulus is taken
   FSTP THETA                 ; principal value is stored, 80287
                              ; stack is popped

; With the test value, THETA is now about 0.2831853
```

SGN — mqerSGN x = (y with x's sign)

Input Parameters

(x) is the top number on the 80287 stack; (y) is the next number on the 80287 stack.

Function

If (x) is greater than or equal to zero, mqerSGN returns the absolute value of (y). If (x) is less than zero, mqerSGN returns the negative of the absolute value of (y).

The positive absolute value of (y) is returned for all values of (x) that are zeroes or pseudo-zeroes, even if (x) is equivalent to −0.

Unnormal values of (x) are legal. If (x) is not a pseudo-zero, only the sign of (x) is relevant to the final answer.

Infinite values of (x) are allowed. The sign of the infinity determines the sign on the answer, even when the 80287 is in projective (unsigned infinity) mode.

Any input (y) is legal, including NaNs, unnormals, denormals, and infinities. The only part of (y) that might be changed upon output is the sign.

Output

The.80287 stack pops once, with the answer replacing the two inputs.

Errors

If (x) is a denormal, the D error is given by an FTST instruction within mqerSGN. If the 80287 is in warning mode (D is masked), mqerSGN uses the denormal to determine the sign of the answer. If the 80287 is in normalizing mode (D is unmasked), the D trap handler will be called with the input still on the 80287 stack. Most trap handlers normalize the argument, reperform the FTST instruction, and continue with the computation of mqerSGN. The trap handler provided by EH287.LIB (described in Chapter 5) replaces the denormal with 0. Thus, the absolute value of (y) will be returned by mqerSGN.

If (x) is a NaN, an I error results. If I is masked, (x) is returned. If I is unmasked, the trap handler is called with the inputs still on the 80287 stack and the 80287 instruction selector set to 264 hex.

Example of PL/M-286 Use

```
mqerSGN: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerSGN;

DECLARE THETA REAL;
DECLARE Y_COOR REAL;
DECLARE PI LITERALLY '3.14159265358979323';

Y_COOR = -0.0000001;  /* Test value */

/* The following code returns either the value PI or -PI.
   If Y_COOR is positive, it returns PI. If Y_COOR is
   negative, it returns -PI. */
```

```
THETA = mqerSGN(PI,Y_COOR);

/* With the test value, THETA now is -PI. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerSGN: FAR

THETA           DQ   ?
Y_COOR          DQ   -0.001    ; initialized to a test value

; The following code returns PI with the sign of Y_COOR,
; just as in the PL/M example above.

   FLDPI              ; first parameter PI onto 80287 stack
   FLD Y_COOR         ; second parameter Y_COOR onto 80287
                      ; stack
   CALL mqerSGN       ; combine sign of Y_COOR with magnitude
                      ; of PI
   FSTP THETA         ; store answer and pop the 80287 stack

; With the test case, THETA is now PI.
```

SIN — mqerSIN x = sine(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerSIN returns the trigonometric sine of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals whose value is less than 2^{-63} return the input value.

Output

The answer replaces the input on the 80287 stack.

Errors

An I error is given for input infinities and NaNs. An I error is also given for unnormals that do not represent values less than 2^{-63}.

If I is unmasked, the trap handler is called with the input still on the stack and the 80287 instruction selector set to 171 hex. If I is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

Example of PL/M-286 Use

```
mqerSIN: PROCEDURE (THETA) REAL EXTERNAL;
   DECLARE THETA REAL;
END mqerSIN;

DECLARE (POLAR_R, POLAR_THETA) REAL;
DECLARE REC_Y REAL;
DECLARE PI LITERALLY '3.14159265358979';
DECLARE DEG_TO_RAD LITERALLY 'PI/180.';

POLAR_R = 2.; POLAR_THETA = 30.;    /* Test values */

/* The following line computes the Y-coordinate of a
   polar-to-rectangular conversion. The input angle is in
   degrees, so it must be converted to radians. */

REC_Y = POLAR_R * mqerSIN(POLAR_THETA * DEG_TO_RAD);

/* Now in the test case, REC_Y = 1. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerSIN: FAR

POLAR_THETA     DQ   30.0
POLAR_R         DQ   2.0
                     ; the above initializations are test
                     ; values.
```

```
REC_Y           DQ   ?
DEG_TO_RAD      DT   3FF98EFA351294E9C8AER    ; the constant
                                              ; PI/180.

; The following lines compute the Y-coordinate of a
; polar-to-rectangular conversion, as in the PL/M
; example above; except that the variables are
; LONG_REAL.

FLD POLAR_THETA       ; degrees angle onto 80287 stack
FLD DEG_TO_RAD
FMUL                  ; converted to radians
CALL mqerSIN          ; sine is taken
FMUL POLAR_R          ; answer scaled to correct radius
FSTP REC_Y            ; Y-coordinate stored and stack is
                      ; popped

; With the test case, REC_Y is now 1.
```

SNH — mqerSNH x = hyperbolic sine(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerSNH returns the hyperbolic sine of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals whose value is less than 2^{-63} return the input value.

Infinite inputs are legal, and return the input value.

Output

The answer replaces the input on the 80287 stack.

Errors

An I error is given for input NaNs. An I error is also given for unnormals that do not represent values less than 2^{-63}.

If I is unmasked, the trap handler is called with the input still on the stack. If I is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

mqerSNH gives an O overflow error if the input is greater than approximately 11355. When O is masked, the value +INFINITY is returned. Likewise, O is given for inputs less than about −11355, with −INFINITY returned for masked O. When O is unmasked, the trap handler is called with the input still on the 80287 stack.

When either trap handler is called, mqerSNH first sets the 80287 instruction selector to 16E hex.

Example of PL/M-286 Use

```
mqerSNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerSNH;

DECLARE (INPUT_VALUE, OUTPUT_VALUE) REAL;

INPUT_VALUE = 2.7;    /* Test value */

OUTPUT_VALUE = mqerSNH(INPUT_VALUE);

/* Now with the test input, OUTPUT_VALUE is about
    14.812526 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerSNH: FAR
```

```
INPUT_VALUE       DQ   -2.7     ; initialization is a test
                                ; value
OUTPUT_VALUE      DQ   ?

   ; The following code duplicates the above PL/M
   ; assignment statement,
   ; except with LONG_REAL variables.

   FLD INPUT_VALUE            ; load the parameter onto the 80287
                              ; stack
   CALL mqerSNH               ; take the hyperbolic sine
   FSTP OUTPUT_VALUE          ; store the answer and pop the
                              ; 80287 stack

; With the test input, OUTPUT_VALUE is now about
; -14.812526
```

TAN — mqerTAN x = tangent(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerTAN returns the trigonometric tangent of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals whose value is less than 2^{-63} return the input value.

Output

The answer replaces the input on the 80287 stack.

Errors

An I error is given for input infinities and NaN's. An I error is also given for unnormals that do not represent values less than 2^{-63}.

If I is unmasked, the trap handler is called with the input still on the stack and the 80287 instruction selector set to 173 hex. If I is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for other invalid inputs.

A Z error is given when the input number is an exact odd multiple of the closest TEMP_REAL number to $\pi/2$. When Z is masked, +INFINITY is returned. When Z is unmasked, the trap handler is called with the input still on the 80287 stack and the 80287 instruction selector set to 173 hex.

Example of PL/M-286 Use

```
mqerTAN: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTAN;

DECLARE PI LITERALLY '3.14159265358979';
DECLARE DEG_TO_RAD LITERALLY 'PI/180.';
DECLARE THETA_DEGREES REAL;
DECLARE SLOPE REAL;

THETA_DEGREES = 135.0;    /* Test value */

/* The following line computes the tangent of the angle
   THETA_DEGREES. The answer is called SLOPE because it is
   the slope of a line which is displaced by THETA_DEGREES
   from the X-axis. */

SLOPE = mqerTAN(THETA_DEGREES * DEG_TO_RAD);

/* Now with the test value, SLOPE = -1. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerTAN: FAR

THETA_DEGREES      DQ   45.00       ; initialization is a test
                                    ; value
SLOPE              DQ   ?
DEG_TO_RAD      DT   3FF98EFA351294E9C8AER    ; the constant
                                              ; PI/180.

    ; The following code computes the tangent just as in
    ; the PL/M example above, except with LONG_REAL
    ; variables.

    FLD THETA_DEGREES        ; load the first parameter onto
                             ; the 80287 stack
    FLD DEG_TO_RAD
    FMUL                     ; convert from degrees to radians
    CALL mqerTAN             ; take the tangent of the radians
                             ; value
    FSTP SLOPE               ; store the answer and pop the
                             ; 80287 stack

; With the test input, SLOPE is now 1.00
```

TNH — mqerTNH x = hyperbolic tangent(x)

Input Parameter

(x) is the top number on the 80287 stack.

Function

mqerTNH returns the hyperbolic tangent of x, where x is an angle expressed in radians. All input zeroes, pseudo-zeroes, and denormals return the input value. Also, unnormals whose value is less than 2^{-63} return the input value.

Infinite inputs are allowed. Input +INFINITY results in +1; −INFINITY results in −1. The sign of the infinity is significant even if the 80287 is in projective (unsigned infinity) mode.

Output

The answer replaces the input on the 80287 stack.

Errors

An I error is given for input NaNs. An I error is also given for unnormals that do not represent values less than 2^{-63}.

If I is unmasked, the trap handler is called with the input still on the stack and the 80287 instruction selector set to 170 hex. If I is masked, the answer is the input for NaNs; the answer is the value INDEFINITE for illegal unnormals.

Example of PL/M-286 Use

```
mqerTNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTNH;

DECLARE (INPUT_VALUE, OUTPUT_VALUE) REAL;

INPUT_VALUE = 0.62;    /* Test value */

OUTPUT_VALUE = mqerTNH(INPUT_VALUE);

/* Now with the test input, OUTPUT_VALUE is about
   0.55112803 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerTNH: FAR

INPUT_VALUE      DQ   -0.62    ; initialization is a test
                               ; value
OUTPUT_VALUE     DQ   ?
```

```
    ; The following code duplicates the above PL/M
    ; assignment statement, except with LONG_REAL
    ; variables.

    FLD INPUT_VALUE         ; load the parameter onto the 80287
                            ; stack
    CALL mqerTNH            ; take the hyperbolic tangent
    FSTP OUTPUT_VALUE       ; store the answer and pop the
                            ; 80287 stack

; With the test input, OUTPUT_VALUE is now about
; -0.55112803
```

Y2X — mqerY2X $x = y^x$

Input Parameters

(x) is the top number on the 80287 stack; (y) is the next number on the 80287 stack.

Function

mqerY2X computes (y) to the (x) power; neither (y) nor (x) is required to be an integer. For most inputs, the formula used is $2^{(x*LG2(y))}$.

The base (y) must be positive for the logarithmic formula to have any meaning. In some cases, however, (y) is allowed to be nonpositive.

- If (x) is positive, a zero (y) gives a zero answer, with no error.
- If (x) is zero, (y) can be negative; the answer is 1.
- If (x) is an integer, (y) can be negative. The function is evaluated using the absolute value of (y). The result is the answer if (x) is even; it is the negative of the answer if (x) is odd.

Zero, infinite, unnormal, or denormal inputs are accepted under certain conditions.

When either input is denormal, it is replaced with an alternate value. The value selected depends on the setting of the 80287 D masking bit. If the 80287 is in normalizing mode (D is unmasked), the input is replaced by 0. If the 80287 is in warning mode (D is masked), the input is replaced by the equivalent unnormal. Note that even though mqerY2X references the D masking bit, it never gives a D error, and it never calls the D trap handler.

An unnormal (y) input is legal only if (x) is a normal integer or infinite value. If (x) is infinite, the function is evaluated as if (y) were the equivalent normal value. If (x) is zero, (y) must be nonzero; in that case, the answer is 1. If (x) is any other integer, it must fit into 32 bits; in that case, the function mqerYI4 is called to obtain the answer.

An unnormal (x) input is legal only if it is nonzero and (y) is infinite or zero. In those cases, (x) is replaced by its normal equivalent.

When the 80287 is in affine (signed infinity) mode, mqerY2X allows infinite inputs in a number of cases:

- If (y) is −INFINITY, (x) must be a nonzero integer. The magnitude of the answer is then INFINITY if (x) is positive; zero if (x) is negative. The sign of the answer is positive if (x) is even; negative if (x) is odd.
- If (y) is +INFINITY, any nonzero (x) is legal. The answer is +INFINITY if (x) is positive; zero if (x) is negative.
- If (x) is +INFINITY, (y) must be positive or any zero, and not equal to 1. The answer is zero if (y) is less than 1, and +INFINITY if (y) is greater than 1.
- If (x) is −INFINITY, (y) must likewise be positive or any zero, and not equal to 1. The answer is +INFINITY if (y) is less than 1, and −INFINITY if (y) is greater than 1.

When the 80287 chip is in projective (unsigned infinity) mode, only one case exists in which any infinite input is allowed: when (y) is infinite, and (x) is a nonzero integer. The result is the same as if the 80287 were in affine mode.

Output

The 80287 stack pops once, with the answer replacing the two inputs.

Errors

mqerY2X first checks for NaN inputs. If either input is a NaN, an I error is given. If I is masked, the input NaN is returned (the larger NaN is returned if both inputs are NaNs).

The legal cases involving unnormal inputs, infinite inputs, and negative (y) inputs are described above. Illegal cases yield an I error. If I is masked, the value INDEFINITE is returned. The case (y = 1) and (x infinite), among others, falls into this category.

It is illegal for both (x) and (y) to have zero values. This too gives an I error, with an INDEFINITE answer if I is masked.

If (y) is any zero and (x) is any negative value (including negative infinity), a Z error is given. If Z is masked, the value +INFINITY is returned.

The O overflow or U underflow error occurs when (y^x) cannot be represented by the TEMP_REAL exponent. If O is masked, an overflow will return the answer +INFINITY. In underflow cases, if U is masked, the correct answer is given if it can be represented by a denormal; otherwise, 0 is given.

All of the errors (I, Z, O, and U) cause the trap handler to be called when the corresponding exception bit is unmasked. mqerY2X leaves the input numbers on the 80287 stack and places the value 26A hex into the 80287 instruction selector before calling the trap handler.

Example of PL/M-286 Use

```
mqerY2X: PROCEDURE (Y,X) REAL EXTERNAL;
  DECLARE (Y,X) REAL;
END mqerY2X;

DECLARE CUBE_ROOT REAL;
DECLARE INPUT_VALUE REAL;

INPUT_VALUE = 17.00;     /* Test value */

/* The following line takes the cube root of the positive
   value INPUT_VALUE. */

CUBE_ROOT = mqerY2X(INPUT_VALUE, 1./3.);

/* With the test input, INPUT_VALUE is now about
   2.5712816 */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerY2X: FAR

INPUT_VALUE      DQ   64.00     ; initialization is a test
                                ; value
```

```
CUBE_ROOT        DQ   ?
ONE_THIRD        DT   0.33333333333333333333   ; constant
                                               ; used below

; The following lines take the cube root just as in the
; PL/M example above, except with LONG_REAL variables.

   FLD INPUT_VALUE              ; load first parameter onto
                                ; 80287 stack
   FLD ONE_THIRD                ; load second parameter onto
                                ; 80287 stack
   CALL mqerY2X                 ; exponentiate
   FSTP CUBE_ROOT               ; store the answer and pop the
                                ; 80287 stack

; With the test input, CUBE_ROOT is now about 4.00
```

YI2 — mqerYI2 x = x ** AX

Input Parameters

(x) is the top number on the 80287 stack. The power to which (x) is raised is the 80286 AX register, interpreted as a twos' complement signed integer.

Function

mqerYI2 raises the real input (x) to an integer power. If the integer is zero, the answer is 1. In this case no error is given, even if (x) is a NaN.

If AX is not zero, the input (x) is first checked for unusual values.

If (x) is +INFINITY, the answer is +INFINITY for positive AX; +0 for negative AX. No error exists.

If (x) is −INFINITY, the magnitude of the answer is INFINITY for positive AX; 0 for negative AX. The sign of the answer is positive for even AX; negative for odd AX.

Zero (x) input is legal if AX is positive. The answer is −0 if (x) is −0 and AX is odd; the answer is +0 in all other cases.

If (x) is denormal, no error is given. However, the 80287 D error masking bit is checked to see what action to take. If the 80287 is in normalizing mode (D is unmasked), (x) is replaced by zero. If the 80287 is in warning mode (D is masked), (x) is replaced by the unnormal number that has the same numeric value as the denormal. The evaluation of mqerYI2 proceeds with the new (x).

If (x) is unnormal and AX is negative, (1/x) is computed, preserving the number of unnormalization bits. Then, the positive power is computed by successively squaring and multiplying by (x).

If (x) is unnormal and AX is positive, the power is computed by successively squaring and multiplying by (x).

For normal, nonzero values of (x), computation of the power proceeds according to the value of the integer power AX.

If the integer power is 64 or greater, or −64 or less, the answer is computed with logarithms. The answer is 2 ** (AX * LG2(x)).

If the integer power is from 1 to 63, the answer is computed by successively squaring and multiplying by (x) to achieve the correct power.

If the integer power is from −63 to −1, mqerYI2 determines if any exceptions would occur if the expression 1 / (x * x * ... x) were evaluated. If not, the expression is evaluated and the answer is returned. If so, the expression (1/x) * (1/x) * ... * (1/x) is evaluated. If the second expression causes exceptions, the trap handler is called.

The maximum number of multiplications performed for any of the above squaring-and-multiplying algorithms is 9.

Output

The answer replaces the input (x) on the 80287 stack. The AX input is destroyed, as allowed by PL/M-286 procedure conventions.

Errors

As stated above, no errors can exist if AX is 0. Otherwise, errors occur in the following cases:

- If (x) is a NaN, an I error is given. If I is masked, the input NaN is returned as the answer.
- A U underflow error occurs when the computed answer is too close to zero to be represented by the exponent. If U is masked, the answer is replaced by the equivalent denormal if it exists; 0 otherwise.
- An O overflow error occurs when the magnitude of the answer is too great to be represented by the exponent. If O is masked, the answer is INFINITY, with the appropriate sign.
- If any of the errors (I, O, or U) occurs with the error unmasked, the trap handler is called. Before calling the trap handler, mqerYI2 sets the 80287 instruction selector to 27C hex and leaves the inputs on the 80287 stack. The integer power is converted to TEMP_REAL and pushed onto the top of the 80287 stack. The base (the original (x)) becomes the second stack element.

Example of PL/M-286 Use

For PL/M-286, the 16-bit input parameter should be on the 80286 stack rather than in the AX register. Therefore, use mqerYIS (instead of mqerYI2) in PL/M-286 programs.

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerYI2: FAR

POWER           DW   9     ; exponent which will be used
REAL_BASE       DQ   1.3   ; number which will be raised to
                           ; POWER
                   ; the above initializations are test
                   ; values
REAL_OUTPUT     DQ   ?

; The following code multiplies POWER copies of REAL_BASE
; together, and stores the answer in REAL_OUTPUT.

   FLD REAL_BASE          ; base parameter goes onto the 80287
                          ; stack
   MOV AX,POWER           ; exponent goes into the AX
                          ; register
   CALL mqerYI2           ; REAL_BASE ** POWER is now on 80287
                          ; stack
   FSTP REAL_OUTPUT       ; store the answer and pop the 80287
                          ; stack

; With the test inputs, REAL_OUTPUT is now about 10.604499
```

YI4 — mqerYI4 x = x ** DXAX

Input Parameters

(x) is the top number on the 80287 stack. The power to which (x) is raised is a 32-bit two's complement value in the 80286 DX and AX registers. DX is the most significant half, and AX is the least significant half.

Function

mqerYI2 raises the real input (x) to an integer power. The input integer is presented in a 32-bit format. Note, however, that 32 bits are rarely necessary to represent an integer exponent. Raising a number to a power greater than 32768 rarely gives meaningful results. Thus mqerYI2 is sufficient for almost every application in which mqerYI4 might be used. mqerYI4 is provided mainly for compatibility with the 32-bit integer types found in Pascal-286 and FORTRAN-286.

If the input integer power is zero, the answer is 1 no matter what the value of (x). In this case no error is given, even if (x) is a NaN.

If DXAX is not zero, the input (x) is first checked for unusual values.

If (x) is +INFINITY, the answer is +INFINITY for positive DXAX; +0 for negative DXAX. No error occurs.

If (x) is −INFINITY, the magnitude of the answer is INFINITY for positive DXAX; 0 for negative DXAX. The sign of the answer is positive for even DXAX; negative for odd DXAX.

Zero (x) input is legal if DXAX is positive. The answer is −0 if (x) is −0 and DXAX is odd; the answer is +0 in all other cases.

If (x) is denormal, no error is given. However, the 80287 D error masking bit is checked to see what action to take. If the 80287 is in normalizing mode (D is unmasked), (x) is replaced by zero. If the 80287 is in warning mode (D is masked), (x) is replaced by the unnormal number that has the same numeric value as the denormal. The evaluation of mqerYI2 proceeds with the new (x).

If (x) is unnormal and DXAX is negative, (1/x) is computed, preserving the number of unnormalization bits. Then, the positive power is computed by successively squaring and multiplying by (x).

If (x) is unnormal and DXAX is positive, the power is computed by successively squaring and multiplying by (x).

For normal, nonzero values of (x), computation of the power proceeds according to the value of the integer power DXAX.

If the integer power is 64 or greater, or −64 or less, the answer is computed with logarithms. The answer is 2 ** (DXAX * LG2(x)).

If the integer power is from 1 to 63, the answer is computed by successively squaring and multiplying by (x) to achieve the correct power.

If the integer power is from −63 to −1, mqerYI4 determines if any exceptions would occur if the expression 1 / (x * x * ... x) were evaluated. If not, the expression is evaluated and the answer is returned. If so, the expression (1/x) * (1/x) * ... * (1/x) is evaluated. If the second expression causes exceptions, the trap handler is called.

The maximum number of multiplications performed for any of the above squaring-and-multiplying algorithms is 9.

Output

The answer replaces the input (x) on the 80287 stack. The DXAX input is destroyed, as allowed by PL/M-286 procedure conventions.

Errors

As stated above, no errors can occur if DXAX is 0. Otherwise, errors occur in the following cases:

- If (x) is a NaN, an I error is given. If I is masked, the input NaN is returned as the answer.
- A U underflow error occurs when the computed answer is too close to zero to be represented by the exponent. If U is masked, the answer is replaced by the equivalent denormal if it exists; 0 otherwise.
- A O overflow error occurs when the magnitude of the answer is too great to be represented by the exponent. If O is masked, the answer is INFINITY with the appropriate sign.
- If any of the errors (I, O, or U) occur with the error unmasked, the trap handler is called. Before calling the trap handler, mqerYI2 sets the 80287 instruction selector to 27C hex and leaves the inputs on the 80287 stack. The integer power is converted to TEMP_REAL and pushed onto the top of the 80287 stack. The base (the original (x)) becomes the second stack element.

Example of PL/M-286 Use

Since PL/M-286 supports no 32-bit integer data type, mqerYI4 cannot be used by PL/M programs. You should use either mqerYIS or mqerY2X.

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerYI4: FAR

POWER           DD   9      ; exponent which will be used
REAL_BASE       DQ   1.3    ; number which will be raised to
                            ; POWER
                            ; the above initializations are
                            ; test values
REAL_OUTPUT     DQ   ?

; The following code multiplies POWER copies of REAL_BASE
; together, and stores the answer in REAL_OUTPUT.

   FLD REAL_BASE                  ; base parameter goes onto
                                  ; the 80287 stack
   MOV AX, WORD PTR POWER         ; low 16 bits of exponent
                                  ; to AX
   MOV DX, WORD PTR (POWER+2)     ; ---high 16 bits to DX
   CALL mqerYI4                   ; REAL_BASE ** POWER is
                                  ; now on 80287 stack
   FSTP REAL_OUTPUT               ; store the answer and pop
                                  ; the 80287 stack

; With the test inputs, REAL_OUTPUT is now about 10.604499
```

YIS — mqerYIS x = x ** STK

Input Parameters

(x) is the top number on the 80287 stack. The power STK to which (x) is to be raised is a 16-bit two's complement integer that is pushed onto the stack before mqerYIS is called.

Function

mqerYIS raises the real input (x) to an integer power. If the integer is zero, the answer is 1. In this case no error is given, even if (x) is a NaN.

If STK is not zero, the input (x) is first checked for unusual values.

If (x) is +INFINITY, the answer is +INFINITY for positive STK; +0 for negative STK. No error occurs.

If (x) is −INFINITY, the magnitude of the answer is INFINITY for positive STK; 0 for negative STK. The sign of the answer is positive for even STK; negative for odd STK.

Zero (x) input is legal if STK is positive. The answer is −0 if (x) is −0 and STK is odd; the answer is +0 in all other cases.

If (x) is denormal, no error is given. However, the 80287 D error masking bit is checked to see what action to take. If the 80287 is in normalizing mode (D is unmasked), (x) is replaced by zero. If the 80287 is in warning mode (D is masked), (x) is replaced by the unnormal number that has the same numeric value as the denormal. The evaluation of mqerYIS proceeds with the new (x).

If (x) is unnormal and STK is negative, (1/x) is computed, preserving the number of unnormalization bits. Then, the positive power is computed by successively squaring and multiplying by (x).

If (x) is unnormal and STK is positive, then the power is computed by successively squaring and multiplying by (x).

For normal, nonzero values of (x), power computation proceeds according to the value of the integer power STK.

If the integer power is 64 or greater, or −64 or less, the answer is computed with logarithms. The answer is 2 ** (STK * LG2(x)).

If the integer power is from 1 to 63, the answer is computed by successively squaring and multiplying by (x) to achieve the correct power.

If the integer power is from −63 to −1, mqerYIS determines if any exceptions would occur if the expression 1 / (x * x * ... x) were evaluated. If not, the expression is evaluated and the answer is returned. If so, expression (1/x) * (1/x) * ... * (1/x) is evaluated. If the second expression causes exceptions, the trap handler is called.

The maximum number of multiplications performed for any of the above squaring-and-multiplying algorithms is 9.

Output

The answer replaces the input (x) on the 80287 stack. mqerYIS returns with the value STK popped off the 80286 stack, so it no longer exists.

Errors

As stated above, no errors can occur if STK is 0. Otherwise, errors occur in the following cases:

- If (x) is a NaN, an I error is given. If I is masked, the input NaN is returned as the answer.
- A U underflow error occurs when the computed answer is too close to zero to be represented by the exponent. If U is masked, the answer is replaced by the equivalent denormal if it exists; the answer is 0 otherwise.
- An O overflow error occurs when the magnitude of the answer is too great to be represented by the exponent. If O is masked, the answer is INFINITY, with the appropriate sign.
- If any of the errors (I, O, or U) occurs with the error unmasked, the trap handler is called. Before calling the trap handler, mqerYIS sets the 80287 instruction selector to 27C hex and leaves the inputs on the 80287 stack. The integer power is converted to TEMP_REAL and pushed onto the top of the 80287 stack. The base (the original (x)) becomes the second stack element.

Example of PL/M-286 Use

```
mqerYIS: PROCEDURE (Y,I) REAL EXTERNAL;
  DECLARE Y REAL, I INTEGER;
END mqerYIS;

DECLARE INTEREST_RATE REAL;
DECLARE NUMBER_OF_PERIODS INTEGER;
DECLARE START_AMOUNT REAL;
DECLARE FINISH_AMOUNT REAL;

INTEREST_RATE = 0.015;       /* Test value */
NUMBER_OF_PERIODS = 12;      /* Test value */
START_AMOUNT = 1000.00;      /* Test value */

/* The following line calculates compound interest for the
   given NUMBER_OF_PERIODS, given the rate INTEREST_RATE
   for each period. INTEREST_RATE is presented as a
   fraction of 1; for example, the value 0.015 represents
   1.5 percent interest for each time period. */

FINISH_AMOUNT = START_AMOUNT* mqerYIS
                 (1.+INTEREST_RATE, NUMBER_OF_PERIODS);

/* With the test inputs, FINISH_AMOUNT is now about
   1195.62. This is the balance of an unpaid loan of
   $1000.00 after one year, if the loan accumulates 1.5
   percent interest every month. */
```

Example of ASM286 Use

```
; This EXTRN must appear outside of all SEGMENT-ENDS
; pairs:
EXTRN mqerYIS: FAR
```

```
INTEREST_RATE         DQ   0.015
NUMBER_OF_PERIODS     DW   12
START_AMOUNT          DQ   1000.00
                         ; the above initializations are test
                         ; values
FINISH_AMOUNT         DQ   ?
; The following code implements the above PL/M example in
; assembly language, with LONG_REAL variables.

   FLD1                       ; 1 onto 80287 stack
   FADD INTEREST_RATE         ; (1 + I) is on stack
   PUSH NUMBER_OF_PERIODS     ; exponent parameter goes onto
                              ; 80286 stack
   CALL mqerYIS               ; (1 + I) ** N is on 80287 stack
   FMUL START_AMOUNT          ; scaled up by the amount of
                              ; money
   FSTP FINISH_AMOUNT         ; store result and pop the 80287
                              ; stack
; NOTE: Do not explicitly POP the NUMBER_OF_PERIODS from
; the 80286 stack -- mqerYIS does that for you.
```

Binding CEL287.LIB to Program Modules

The final action to take to use CEL287.LIB in your programs is to include the file name CEL287.LIB into the appropriate BND286 command. You must also bind in 80287.LIB.

If you are also using EH287.LIB, you must give the name EH287.LIB after CEL287.LIB. If you put EH287.LIB before CEL287.LIB, the program will link with no error messages, but it will halt after the first CEL287 function is called.

Following is the suggested order for object modules in your BND286 statement:

Your object modules
DC287.LIB (if you are using it)
CEL287.LIB
EH287.LIB (if you are using it)
80287.LIB

For example, if you are binding your PL/M-286 modules MYMOD1.OBJ and MYMOD2.OBJ into a program using the error handler, issue the following command:

```
-BND286 :F1:MYMOD1.OBJ, &<cr>
        :F1:MYMOD2.OBJ, &<cr>
        :F0:CEL287.LIB, &<cr>
        :F0:EH287.LIB, &<cr>
        :F0:80287.LIB &<cr>
        OBJECT (:F1:MYPROG.LNK)<cr>
```

If you have a single ASM286-generated object module :F1:MYMOD1.OBJ to be executed, issue the following command:

```
-BND286 :F1:MYPROG.OBJ, &<cr>
        :F0:CEL287.LIB, &<cr>
        :F0:80287.LIB &<cr>
        OBJECT (:F1:MYPROG.LNK)<cr>
```

CHAPTER 5
THE ERROR HANDLER MODULE

Overview

This chapter describes EH287.LIB, a library of five utility procedures that you can use to write trap handlers. Trap handlers are procedures that are called when an unmasked 80287 error occurs.

EH287.LIB also contains a set of "dummy" public symbols for all the CEL287.LIB functions. These symbols save you code when you do not use all the CEL287.LIB functions. Their presence, however, makes it absolutely necessary that you link EH287.LIB and CEL287.LIB in the correct order. The last section of this chapter tells you how to do so.

EH287.LIB also contains a set of alternate public names for its procedures, which are used by some Intel translators. They are listed in Appendix F.

The 80287 error reporting mechanism can be used not only to report error conditions, but also to let software implement modes and functions not directly supported by the chip. This chapter defines three such extensions to the 80287: normalizing mode, nontrapping NaNs, and nonordered comparison. The utility procedures support these extra features.

DECODE is called near the beginning of the trap handler. It preserves the complete state of the 80287 and also identifies what function called the trap handler, with what arguments and/or results. DECODE eliminates much of the effort needed to determine what error caused the trap handler to be called.

NORMAL provides the "normalizing mode" capability for handling the D exception (described in the following section). By calling NORMAL in your trap handler, you eliminate the need to write code that tests for nonnormal inputs.

SIEVE provides two capabilities for handling the I exception. It implements nontrapping NaNs and nonordered comparisons (both described in the following sections). These two IEEE standard features reduce the incidence of multiple error reports for a single bad input.

ENCODE is called near the end of the trap handler. It restores the state of the 80287 saved by DECODE and performs a choice of concluding actions either by retrying the offending function or returning a specified result. DC287 exceptions cannot be retried. ENCODE provides a common path for exiting the trap handler and resuming execution of the user program.

FILTER calls each of the above four procedures. If your error handler does nothing more than detect fatal errors and implement the features supported by SIEVE and NORMAL, your interface to EH287.LIB can be accomplished with a single call to FILTER.

Normalizing Mode

Normalizing mode allows you to perform floating-point operations without having to worry about whether the operands are in normal form. All denormal inputs will be normalized before the operation, without any user intervention.

The 80287 provides the D error, which warns you that an operand is not normal. You can implement normalizing mode in software by unmasking the D error and

providing a D trap handler. The handler should perform the needed normalization and then retry the operation. NORMAL, provided in the 80287 support library, gives normalizations adequate for a majority of applications.

Nontrapping NaNs

The large number of representations for NaNs gives you the chance to put diagnostic information into a NaN. The information can be passed along as the NaN undergoes multiple floating-point operations. The information can not only tell where the NaN came from, but can also control further error action.

EH287.LIB adopts the convention that the top bit of the fractional part of a NaN is a control bit used as follows: if the bit is 1, the NaN is considered a nontrapping NaN, for which no further error need be explicitly reported. You can return a nontrapping NaN as the result of an invalid operation. Then, when the NaN passes through more arithmetic, no more errors will be reported. You thus avoid multiple error messages which really come from only one error.

The 80287 does not distinguish between trapping and nontrapping NaNs. All NaNs generate an I error when they are used. However, you can provide an I trap handler, which distinguishes between trapping and nontrapping NaNs. The procedure SIEVE does this for you.

Nonordered Comparisons

When testing two floating-point numbers for equality, you may or may not want an error to be reported if those numbers are NaNs. The 80287 provides the FCOM and FTST instructions, which report an I error if they are given a NaN input.

To suppress error reporting for NaNs in FCOM and FTST, the following convention is recommended: if either FCOM or FTST is followed by a MOV AX,AX instruction (8BC0 hex), the I trap handler should treat nontrapping NaNs as legal inputs. It should return the answer NONORDERED (C3 = C0 = 1), even if the two inputs are the same NaN, and act as if no I error had occurred. The procedure SIEVE follows this suggested convention.

Note that comparisons coded in PL/M-286 generate FCOM and FTST instructions, which are not followed by a MOV AX,AX instruction. Therefore, according to the EH287.LIB convention, PL/M-286 comparisons of nontrapping NaNs are not considered legal. No way exists to cause PL/M-286 to insert a MOV AX,AX instruction after a comparison.

The ESTATE287 Data Structure

ESTATE287 is a 144-byte data structure created by DECODE and used by the other EH287.LIB utility procedures. It contains most of the information your trap handler needs to provide customized error recovery: the state of the 80287, the identity of the offending operation, the values and formats of the operands, and possible already-calculated results.

You will typically receive an ESTATE287 structure from DECODE and pass it back to ENCODE mostly unchanged. You need not become familiar with those parts of ESTATE287 that you do not change.

Following is a description of each of the fields of the ESTATE287 structure. The offsets mentioned are decimal numbers that give the number of bytes from the beginning of ESTATE287 to the beginning of the field.

OPERATION is a WORD, offset 0, that contains an error code identifies which procedure or 80287 instruction caused the error. Appendix D gives the error codes for 80287 instructions. The error codes for CEL287 functions are the last two digits of the codes in Appendix E. The error code for all DC287 functions is 0C8 hex.

ARGUMENT is a BYTE, offset 2, that identifies the types and locations of the arguments of the interrupted operation. See figure 5-1 for the layout of the bit fields within the ARGUMENT byte. The 3-bit fields ATYPE1 and ATYPE2 indicate the types of ARG1 and ARG2, according to the following codes:

0 no operand
1 top of 80287 stack: ST(0)
2 next element on 80287 stack: ST(1)
3 the element of the 80287 stack specified by REGISTER
4 a number in 80286 memory of a type given by FORMAT
5 a TEMP_REAL operand
6 a 64-bit integer operand
7 a binary-coded decimal operand

The PUSH ONCE bit is 1 if the result is pushed onto the 80287 stack (rather than replacing one or two of the input arguments).

For example, ARGUMENT would equal 21 hex if the instruction FPREM caused the error, because the arguments to FPREM are the top two elements on the 80287 stack, and the result replaces the inputs rather than causing the 80287 stack to be pushed.

ARG1(5) is a WORD array, offset 3, that gives the first argument of the operation in the format specified by ARGUMENT.

ARG1_FULL is a Boolean BYTE, offset 13, the bottom bit of which is 1 if ARG1 is present. If the error handler is called after the offending operation is done, the argument may no longer exist.

ARG2(5) is a WORD array, offset 14, that gives the second argument of the operation in the format specified by ARGUMENT.

ARG2_FULL is a Boolean BYTE, offset 24, the bottom bit of which is 1 if ARG2 is present. If only one argument exists, or if the error handler is called after the offending operation is done, this argument may not exist.

RESULT is a BYTE, offset 25, that identifies the types and locations of the results of the interrupted operation. See figure 5-2. The 3-bit fields RTYPE1 and RTYPE2 use the same codes as the corresponding fields in the ARGUMENT byte. The POP ONCE bit is set if the operation causes the 80287 stack to pop exactly once; the POP TWICE bit is set if the operation causes the 80287 stack to pop twice (i.e., the operation is FCOMPP.)

7							0
NOT USED		ATYPE 2		PUSH ONCE		ATYPE 1	

Figure 5-1. Bit Fields of the ARGUMENT Byte in ESTATE287 121725-2

7							0
POP TWICE		RTYPE 2		POP ONCE		RTYPE 1	

Figure 5-2. Bit Fields of the RESULT Byte in ESTATE287 121725-3

RES1(5) is a WORD array, offset 26, that gives the first result of the operation in the format specified by RESULT.

RES1_FULL is a Boolean BYTE, offset 36, the bottom bit of which is 1 if RES1 is present. If the error handler is called before the offending operation is done, the result does not yet exist.

RES2(5) is a WORD array, offset 37, that gives the second result of the operation in the format specified by RESULT.

RES2_FULL is a Boolean BYTE, offset 47, bottom bit of which is 1 if RES2 is present. If only one result exists, or if the error handler is called before the offending operation is done, this result does not exist.

FORMAT is a BYTE, offset 48, that specifies the memory data type when a field of ARGUMENT or RESULT has value 4. (Only one such field ever exists.) The possible values of FORMAT are as follows:

0 for SHORT_REAL (32 bits)
1 for a 32-bit integer
2 for LONG_REAL (64 bits)
3 for a 16-bit integer

REGISTER is a BYTE, offset 49, that specifies the 80287 stack element number when a field of ARGUMENT or RESULT has the value 3. (No more than one such field can have this value.) The values of REGISTER range from 0 for the top stack element to 7 from the bottom-most stack element.

SAVE287(47) is a WORD array, offset 50, that contains the state of the 80287 as defined by the 80287 FSAVE instruction. Because DECODE is called after the 80287 exceptions have been cleared, the 80287 status word stored into SAVE87 differs from the status word as it existed when the trap handler was invoked. This former value must be maintained separately from ESTATE287; it appears as the parameter ERRORS287 in all the EH287.LIB routines.

Writing an 80287 Exception Handler in ASM286 Using EH287.LIB

By using EH287.LIB, you eliminate the difficult aspects of interfacing to the 80287 for your error recovery. However, a strict protocol must be followed. Following is a template of coding structure for an 80287 error handler written in ASM286. If your customized error recovery is limited to errors that do not examine the ESTATE287 structure, follow the simpler template given under FILTER.

1. The handler procedure must be an INTERRUPT procedure. To do this you must use the system builder, BLD286.
2. If you expect to return to the code that caused the handler to be called, you must preserve all 80286 registers. The 80286 flags are automatically saved and restored, because this is an interrupt routine.
3. The first floating-point instruction of the handler should be an FNSTSW instruction, which stores the 80287 status word into a 16-bit memory location. This status word is the ERRORS287 parameter given to the EH287.LIB procedures.
4. To ensure synchronization with the 80287 at this point, you must provide an arbitrary access to the 80286 memory space. A PUSH AX instruction followed by a POP AX instruction is recommended.

5. An FNCLEX instruction should follow to clear the 80287 exceptions. The exceptions must be cleared for the following EH287.LIB calls to work properly.

6. The parameters to DECODE should be pushed onto the 80286 stack, and DECODE should be called.

7. If you intend to use NORMAL and SIEVE, their calls should come immediately after the call to DECODE. You should not have intervening code that alters ESTATE287. NORMAL and SIEVE require ESTATE287 to be defined by DECODE. If ESTATE287 has no values that could be output by DECODE, the results of NORMAL and SIEVE are undefined.

8. If you have any customized code in your error handler, it should appear here.

9. If the handler is returning to the calling environment, the parameters to ENCODE should be pushed onto the 80286 stack, and ENCODE should be called.

10. The 80286 registers saved at the beginning of the exception handler should now be restored.

11. The exception handler should be exited using an IRET instruction. A simple RET instruction will not do because the 80286 pushes its flags onto its stack when the interrupt is executed.

You should also remember that if the possibility of a recursive call to the error handler exists (which can happen if ENCODE retries an instruction with exceptions unmasked), all data storage used by your handler must be allocated on the 80286 stack. This includes the ERRORS287 word and the entire ESTATE287 structure.

Example of an 80287 Exception Handler Written in ASM286

```
NAME HANDLER_287

EXTRN DECODE: FAR
EXTRN SIEVE: FAR
EXTRN ENCODE: FAR
EXTRN NORMAL: FAR

CODE SEGMENT ER PUBLIC

I_MASK          EQU    0001H            ; Position of "I"
                                        ; error bit and "I"
                                        ; mask

STACK_LAYOUT STRUC                      ; Pointed at by BP
                                        ; during TRAP_HANDLER

 OPERATION             DW ?             ; ESTATE287 template
                                        ; begins with this
                                        ; line

 ARGUMENT              DB ?
 ARG1                  DW 5 DUP(?)
 ARG1_FULL             DB ?
 ARG2                  DW 5 DUP(?)
 ARG2_FULL             DB ?
 RESULT                DB ?
```

```
RES1                DW 5 DUP(?)
RES1_FULL           DB ?
RES2                DW 5 DUP(?)
RES2_FULL           DB ?
FORMAT              DB ?
REGISTER            DB ?
CONTROL_WORD        DW ?              ; Start of 94-byte
                                      ; FSAVE template
STATUS_WORD         DW ?              ; As it exists after
                                      ; clearing exceptions
TAG_WORD            DW ?
ERROR_POINTERS      DW 4 DUP(?)
STACK287            DT 8 DUP(?)       ; Last line of FSAVE
                                      ; and of ESTATE287
RETRY_CONTROL       DW ?              ; Retry 80287 Control
                                      ; setting for ENCODE
RETRY_FLAG          DB ?              ; Boolean parameter to
                                      ; ENCODE
ERRORS287           DW ?              ; 80287 Status Word
                                      ; before clearing

; You can place additional stack-allocated variables here,
; for your custom code. You refer to them as
; [BP].your_var_name. If the number of bytes of inserted
; space is odd, you should eliminate the following dummy
; variable.

DUMMY_EVEN_ALIGN    DB ?              ; Not used; filler to
                                      ; keep SP even
REGISTERS_286       DW 8 DUP(?)       ; Pushed by
                                      ; PUSH_REGISTERS
FLAGS_286           DW ?              ; Pushed by 80286 when
                                      ; interrupt called
RET_ADDRESS         DD ?              ; From TRAP_HANDLER

STACK_LAYOUT ENDS

; TRAP_HANDLER is a functioning 80287 exception handler
; written in ASM286, using the procedures of EH287.LIB. It
; assumes that the only unmasked exceptions are "I" and
; "D", though we indicate below where you could insert
; code to handle other exceptions.

TRAP_HANDLER PROC FAR

  CALL PUSH_REGISTERS                 ; This interrupt will
                                      ; preserve 80286 registers
  SUB SP,OFFSET REGISTERS_286         ; Allocate room for
                                      ; STACK_LAYOUT
  MOV BP,SP                           ; Set up indexing into
                                      ; STACK_LAYOUT
  FNSTSW [BP].ERRORS287               ; Save the errors that
                                      ; caused the exception
  PUSH AX                             ; Arbitrary memory access
                                      ; to synchronize
  POP AX                              ; with the 80287
  FNCLEX                              ; Clear the exceptions so
                                      ; handler can use 80287
  CALL PUSH_ESTATE_ERRORS             ; Push parameters to
```

```
                                      ; DECODE onto stack
    CALL DECODE                       ; ESTATE287 is now filled
                                      ; with valid data
    MOV AX,[BP].CONTROL_WORD          ; Existing control word is
                                      ; the default for
    MOV [BP].RETRY_CONTROL,AX         ; any ENCODE retry
                                      ; attempt
    MOV [BP].RETRY_FLAG,0FFH          ; Default setting: retry
                                      ; will take place
    CALL PUSH_ESTATE_ERRORS           ; Push parameters to
                                      ; NORMAL onto stack
    CALL NORMAL                       ; Handle "D" errors
    RCR AL,1                          ; Test Boolean answer: was
                                      ; "D" the only error?
    JC ENCODE_EXIT                    ; If so then no other
                                      ; checking is necessary

    OR [BP].RETRY_CONTROL,I_MASK      ; Mask "I" exception
    CALL PUSH_ESTATE_ERRORS           ; Push parameters to SIEVE
    CALL SIEVE                        ; Check for legal non-
                                      ; trapping NaN
    RCR AL,1                          ; Test Boolean answer: was
                                      ; there such a NaN?
    JC ENCODE_EXIT                    ; If so then we can exit;
                                      ; "I" has been cleared

    TEST [BP].ERRORS287,I_MASK        ; Was "I" error bit set?
    JNZ ACTUAL_I_ERROR

    ; Here you could insert code to examine [BP].ERRORS287 to
    ; detect exceptions other than "I" or "D". If those
    ; other exceptions are detected, you can provide your
    ; own code to handle them.

    JMP ENCODE_EXIT

ACTUAL_I_ERROR:

    ; Here you may place your own customized code to deal
    ; with "I" exceptions other than legal nontrapping
    ; NaNs and denormalized inputs. The following lines set
    ; the "I" error bit, mask the "I" exception, and drop
    ; through to ENCODE_EXIT. This simulates the masked "I"
    ; exception, but with nontrapping NaNs implemented.
    ; The user program must unmask the "I" exception when it
    ; tests and clears the "I" error bit.

    OR [BP].STATUS_WORD,I_MASK        ; Set the "I" error bit
    OR [BP].CONTROL_WORD,I_MASK       ; Mask the "I" exception

ENCODE_EXIT:
    CALL PUSH_ESTATE_ERRORS           ; Push first two ENCODE
                                      ; parameters
    PUSH [BP].RETRY_CONTROL           ; Push third ENCODE
                                      ; parameter
    PUSH WORD PTR [BP].RETRY_FLAG     ; Push fourth ENCODE
                                      ; parameter
    CALL ENCODE                       ; Restore post-exception
                                      ; 80287 state
```

```
        ADD SP,OFFSET REGISTERS_86      ; Release the STACK_LAYOUT
                                        ; storage
        CALL POP_REGISTERS              ; Restore the eight 80286
                                        ; registers
        IRET                            ; Restore 80286 flags;
                                        ; long-return to caller

TRAP_HANDLER ENDP

; PUSH_ESTATE_ERRORS causes the two parameters
; ESTATE287_PTR and ERRORS287 to be pushed onto the 80286
; stack, prior to a call to one of the EH287.LIB procedures
; which call for these parameters.

PUSH_ESTATE_ERRORS PROC NEAR
        POP DX                          ; save the return address
        PUSH SS                         ; push the segment half of
                                        ; ESTATE287_PTR
        LEA AX,[BP].OPERATION           ; push the offset half of
                                        ; ESTATE287_PTR
        PUSH AX                         ; push complete
        PUSH [BP].ERRORS287             ; ERRORS287 is the bottom
                                        ; half of this byte
        JMP DX                          ; this is the RETURN from
                                        ; PUSH_ESTATE_ERRORS
PUSH_ESTATE_ERRORS ENDP

; PUSH_REGISTERS causes the eight 80286 registers
; SI,DI,ES,BP,DX,CX,BX,AX to be pushed onto the 80286
; stack. The registers can be popped off with a call to
; POP_REGISTERS.

PUSH_REGISTERS PROC NEAR
        PUSH DI
        PUSH ES
        PUSH BP
        PUSH DX
        PUSH CX
        PUSH BX
        PUSH AX
        MOV BP,SP               ; Get stack pointer into an index
                                ; register
        XCHG SI,[BP+14]         ; Exchange the return address with
                                ; SI which is being saved
        JMP SI                  ; This is the RETURN of the
                                ; procedure
PUSH_REGISTERS ENDP

; POP_REGISTERS causes the eight registers which were
; pushed by PUSH_REGISTERS to be popped, restoring them to
; their original values.

POP_REGISTERS PROC NEAR
        POP SI                  ; Hold this call's return address
                                ; temprarily in SI
        MOV BP,SP               ; Get the stack pointer into an
                                ; index register
        XCHG SI,[BP+14]         ; Restore SI, and position the
                                ; return address
```

```
    POP AX
    POP BX
    POP CX
    POP DX
    POP BP
    POP ES
    POP DI
    RET                          ; Return address is in position due
                                 ; to above XCHG
POP_REGISTERS ENDP

CODE ENDS

END
```

Writing an 80287 Exception Handler in PL/M-286 Using EH287.LIB

Following is a template of coding structure for any 80287 error handler written in PL/M-286. Note that many of the protocols required for an ASM286 error handler are not needed in PL/M-286. They are provided automatically by PL/M-286 and its built-in functions.

If your customized error recovery is limited to errors that do not examine the ESTATE287 structure, you can follow the simpler template given under FILTER.

1. Your error handler procedure must have the attributes INTERRUPT and PUBLIC.
2. You must declare the structure ESTATE287 and the WORD variable ERRORS287.
3. If you use any other variables in your error handler, they should also be declared within the procedure. All data must be declared within the procedure for the error handler to be reentrant.
4. You should make the assignment ERRORS287 = GET$REAL$ERROR.
5. You should call DECODE with the approriate parameters.
6. If you intend to use NORMAL and SIEVE, their calls should come immeditely after the call to DECODE. NORMAL and SIEVE require ESTATE287 to be defined by DECODE. If ESTATE287 has no values that could be output by DECODE, the results of NORMAL and SIEVE are undefined.
7. If you have any customized code in your error handler, it should appear here.
8. If the handler is returning to the calling environment, you should call ENCODE with the appropriate parameters.

Example of an 80287 Exception Handler Written in PL/M-286

```
HANDLER_MODULE: DO;

DECODE: PROCEDURE (ESTATE287_PTR,ERRORS287) EXTERNAL;
  DECLARE ESTATE287_PTR POINTER, ERRORS287 WORD;
END;
```

```
ENCODE: PROCEDURE (ESTATE287_PTR,ERRORS287,CONTROL287,
                    RETRY_FLAG) EXTERNAL;
  DECLARE ESTATE287_PTR POINTER;
  DECLARE ERRORS287 WORD;
  DECLARE CONTROL287 WORD;
  DECLARE RETRY_FLAG BYTE;
END;

NORMAL: PROCEDURE (ESTATE287_PTR,ERRORS287) BYTE EXTERNAL;
  DECLARE ESTATE287_PTR POINTER, ERRORS287 WORD;
END;

SIEVE: PROCEDURE (ESTATE287_PTR,ERRORS287) BYTE EXTERNAL;
  DECLARE ESTATE287_PTR POINTER, ERRORS287 WORD;
END;

/* TRAP_HANDLER is a functioning 80287 exception handler
   written in PL/M-286, using the procedures of EH287.LIB.
   It assumes that the only unmasked exceptions are "I"
   and "D", though we indicate below where you could
   insert code to handle other exceptions. */

DECLARE I$ERROR$BIT LITERALLY '0001H';   /* To set the "I"
                                             error bit */
DECLARE TRUE LITERALLY '0FFH';

TRAP_HANDLER: PROCEDURE INTERRUPT PUBLIC;

  DECLARE ESTATE287 STRUCTURE (
     OPERATION WORD,
     ARGUMENT BYTE,
     ARG1(5) WORD, ARG1_FULL BYTE,
     ARG2(5) WORD, ARG2_FULL BYTE,
     RESULT BYTE,
     RES1(5) WORD, RES1_FULL BYTE,
     RES2(5) WORD, RES2_FULL BYTE,
     FORMAT BYTE,
     REGISTER BYTE,
     CONTROL_WORD WORD,
     STATUS_WORD WORD,
     TAG_WORD WORD,
     ERROR_POINTERS(4) WORD,
     STACK_87(40) WORD);
  DECLARE ERRORS287 WORD;
  DECLARE CONTROL287 WORD;
  DECLARE RETRY_FLAG BYTE;

  ERRORS287 = GET$REAL$ERROR;
  CALL DECODE(@ESTATE287,ERRORS287);
  CONTROL287 = ESTATE287.CONTROL_WORD;
  RETRY_FLAG = TRUE;

  IF NOT NORMAL(@ESTATE287,ERRORS287)
   THEN DO;
    CONTROL287 = CONTROL287 OR I$ERROR$BIT;
    IF NOT SIEVE(@ESTATE287,ERRORS287)
     THEN DO;
      IF ERRORS287
       THEN DO;
```

```
            /* Here you may place your own customized code to
               deal with "I" exceptions other than non-
               trapping NaNs and denormalized inputs. The
               following lines set the "I" error bit, mask the
               "I" exception, and drop through to the RETRY
               exit. This simulates the masked "I" exception,
               but with nontrapping NaNs implemented. The
               user program must unmask the "I" exception when
               it tests and clears the "I" error bit. */
            ESTATE287.STATUS_WORD = ESTATE287.STATUS_WORD OR
                                    I$ERROR$BIT;
            ESTATE287.CONTROL_WORD = CONTROL287;
            END;
           ELSE DO;
            /* Here you could insert code to examine ERRORS287
               to detect exceptions other than "I" and "D". If
               those other exceptions are detected, you can
               provide your own code. */
            END;
          END;
        END;

      CALL ENCODE(@ESTATE287,ERRORS287,CONTROL287,RETRY_FLAG);

    END TRAP_HANDLER;

    END HANDLER_MODULE;
```

DECODE — Decode trapped operation and save 80287 status

Input

Two parameters, totalling six bytes, must be pushed onto the 80286 stack before calling DECODE.

First, the four-byte pointer ESTATE287_PTR is pushed. As usual, the segment of ESTATE287_PTR is pushed first, followed by the offset. DECODE sends its output to the 144-byte memory buffer pointed to by ESTATE287_PTR.

Second, a two-byte quantity, the bottom byte of which is ERRORS287, is pushed onto the stack. ERRORS287 is the bottom byte of the 80287 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of the two-byte quantity is ignored.

Function

DECODE provides, in a standardized format, all the information an exception handler might need to deal with the error condition. This includes the complete state of the 80287, the identity of the offending operation, and the existence, formats, and values of any arguments or results.

DECODE has programmed into it the error calling specifications of all 80287 instructions and all CEL287.LIB functions. Once DECODE has identified the interrupted operation, it uses these specifications to fill ESTATE287 with the correct arguments and results.

An exception to the programming specifications occurs for the CEL287 functions that return values in 80286 registers. These functions are mqerIA2, mqerIA4, mqerIC2, mqerIC4, mqerIE2, and mqerIE4. The results are specified by DECODE to go to the 80287 stack, not to 80286 registers.

DECODE identifies DC287 errors by setting OPERATION to 0C8 hex, but no information about DC287 arguments or results is available because they reside in a location on the 80286 stack unknown to DECODE.

Note that for FCOMP and FCOMPP instructions with denormal arguments the trap handler is called with the inputs already popped off the 80287 stack. DECODE recovers these arguments from the bottom of the 80287 stack and pushes them back onto the stack top.

Output

All output is performed through the 144-byte memory buffer pointed at by the input parameter ESTATE287_PTR. The fields of the ESTATE287 structure are described in "The ESTATE287 Data Structure" at the beginning of this chapter.

The 80287 itself is left completely cleared to its initialized state. This is the input state expected by the other procedures of EH287.LIB.

PL/M-286 Declaration and Calling Sequence

```
DECODE: PROCEDURE (ESTATE287_PTR,ERRORS287) EXTERNAL;
  DECLARE ESTATE287_PTR POINTER, ERRORS287 WORD;
END;
```

```
/* Assume ESTATE287 is already declared as presented
   earlier in this chapter */

DECLARE ERRORS287 WORD;

/* Assume ERRORS287 has been set to the value of the bottom
   part of the 80287 status word before exceptions were
   cleared. */

CALL DECODE( @ESTATE287, ERRORS287 );
```

ASM286 Declaration and Calling Sequence

```
; the following line must occur outside of all SEGMENT-
; ENDS pairs

      EXTRN DECODE: FAR

ESTATE_TEMPLATE STRUC
    DB   144 DUP(?)        ; The individual fields of ESTATE287
                           ; can be declared
                           ; as in STACK_LAYOUT at the
                           ; beginning of this chapter.
ESTATE_TEMPLATE ENDS

DATA SEGMENT
      ESTATE_287 ESTATE_TEMPLATE
      ERRORS287  DW   ?
DATA ENDS

; the following code assumes that DS contains the segment
; of ESTATE287

      ASSUME DS:DATA
      PUSH DS                        ; push the segment of
                                     ; ESTATE287 onto the stack
      MOV AX,OFFSET(ESTATE287)
      PUSH AX                        ; 4-byte pointer is now
                                     ; pushed onto stack
      MOV AL,ERRORS287               ; second parameter
      PUSH AX                        ; pushed onto stack
      CALL DECODE                    ; ESTATE287 is now filled
                                     ; with information
```

ENCODE — Restore 80287 status and operation environment

Input

ENCODE expects the 80287 to be completely cleared to its initialized state. Information about the 80287 is obtained from ESTATE287, not from the entry state of the 80287 itself.

Four parameters, totalling ten bytes, must be pushed onto the 80286 stack before calling ENCODE. The first parameter is four bytes; each of the other three parameters is two bytes.

The first parameter, ESTATE287_PTR, points to the 144-byte ESTATE287 structure that was filled by a previous call to DECODE and possibly modified by intervening code. ESTATE287_PTR is a four-byte pointer with the segment pushed first and the offset pushed next.

The second parameter pushed onto the stack is ERRORS287. ERRORS287 is a WORD parameter, the bottom byte of which is the bottom byte of the 80287 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of ERRORS287 is ignored.

The third parameter, CONTROL287, is a WORD parameter. It contains the value the caller wants to place into the 80287 control word before retrying the function when RETRY_FLAG is true. If RETRY_FLAG is false, CONTROL287 is ignored.

The fourth and last parameter pushed onto the stack is RETRY_FLAG. RETRY _FLAG is a BYTE parameter, the bottom half of the WORD that is pushed onto the 80286 stack (the top half is ignored). RETRY_FLAG is a Boolean control input that tells ENCODE what action to take after restoring the 80287 environment, as described below.

Function

ENCODE restores the 80287 to a state that represents the completion of the operation that caused an error trap. This is accomplished by restoring the 80287 to the state stored in the ESTATE287 structure, after performing one of two actions, depending on RETRY_FLAG.

If the bottom bit of RETRY_FLAG is 0, the operation is assumed to have already been performed, and the results are assumed to be in ESTATE287. The results are copied to their destination. If insufficient results are available in ESTATE287, the value INDEFINITE may be used.

If the bottom bit of RETRY_FLAG is 1, the operation is identified from ESTATE287 and reperformed using operands as specified in ESTATE287. The parameter CONTROL287 is used as the 80287 control word for the retry; you can thus select which exceptions will be unmasked. After the retry is complete, the 80287 control word as it exists in ESTATE287 is restored.

If the operation being retried is a load or store that merely moves data without transforming it, ENCODE will perform the operation, with exceptions masked, using the ARGUMENT and RESULT information of ESTATE287. This allows you to call ENCODE, with a true RETRY_FLAG in all cases when NORMAL returns TRUE, without fear that the retry will repeat the D error.

Note that ENCODE cannot be called with RETRY_FLAG true if the error came from DC287, because ESTATE287 contains insufficient information about DC287.

For the same reason, ENCODE cannot retry the CEL287 functions mqerIC2, mqerIC4, mqerIE2, or mqerIE4 after a P precision error.

ENCODE can perform the retry for I errors that come from the above four CEL287 functions as well as from mqerIA2 and mqerIA4, but the results will go to the 80287 stack—not to the correct 80286 registers. Your code must identify these functions and pop the answer from the 80287 to the correct destination in order to resume execution of the trapped program.

If your CONTROL287 parameter contains unmasked exceptions, it is possible for an exception handler to be called during the retry and for ENCODE to be invoked recursively. If you unmask the exception that caused the error, you should have modified the original arguments or results to avoid an infinite recursion.

Output

ENCODE returns with the 80287 restored to the appropriate post-exception state, as indicated by ESTATE287 and the effects of either the retry or the movement of results to their destination. Also, the input parameters are popped from the 80286 stack.

PL/M-286 Declaration and Calling Sequence

```
ENCODE: PROCEDURE (ESTATE287_PTR,ERRORS287,CONTROL287,
    RETRY_FLAG) EXTERNAL;
  DECLARE ESTATE287_PTR POINTER;
  DECLARE ERRORS287 WORD;
  DECLARE CONTROL287 WORD;
  DECLARE RETRY_FLAG BYTE;
END;

/* Assume ESTATE287 is already declared as presented
   earlier in this chapter. */

DECLARE ERRORS287 WORD;
DECLARE CONTROL287 WORD;
DECLARE RETRY_FLAG BYTE;

/* Assume that all the parameters have been filled with
   values. */

CALL ENCODE( @ESTATE287, ERRORS287, CONTROL287, RETRY_FLAG );

/* The 80287 has now been returned to an appropriate post-
   exception state. */
```

ASM286 Declaration and Calling Sequence

```
; the following line must occur outside of all SEGMENT-
; ENDS pairs

    EXTRN ENCODE: FAR

DATA SEGMENT
    ESTATE_287   ESTATE_TEMPLATE
    ERRORS287   DW   ?
DATA ENDS
```

```
; The following code assumes that DS contains the segment
; of ESTATE287. It also assumes that the parameters to
; ENCODE have been set to appropriate values.

    ASSUME DS:DATA
    PUSH DS                          ; push the segment of
                                     ; ESTATE287 onto the stack
    MOV AX,OFFSET(ESTATE287)
    PUSH AX                          ; 4-byte pointer is now
                                     ; pushed onto stack
    MOV AX,ERRORS287
    PUSH AX                          ; second parameter is
                                     ; pushed onto stack
    MOV AX,CONTROL287                ; load third parameter
    PUSH AX                          ; push onto stack
    MOV AL, RETRY_FLAG               ; last parameter
    PUSH AX                          ; pushed onto stack
    CALL ENCODE

; the 80287 is now restored to an appropriate post-
; exception state
```

FILTER — Filter denormals and nontrapping NaNs from user error handling

Input

The two-byte quantity, the bottom byte of which is ERRORS287, is pushed onto the 80286 stack before calling FILTER. ERRORS287 is the bottom byte of the 80287 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of the two-byte quantity is ignored.

Function

FILTER provides a single interface to EH287.LIB, which calls all of the other four functions (DECODE, SIEVE, NORMAL, and ENCODE). If ERRORS287 indicates that a D error caused the trap handler to be called, the denormal argument is found and normalized. If an I error caused the trap, FILTER indicates whether the exception was caused by a legal nontrapping NaN.

FILTER is a Boolean function. It returns TRUE if the trap was indeed caused by one of the above-mentioned I or D cases. TRUE means that you may return immediately from the trap handler. FILTER returns FALSE if it has not handled the error. If FILTER returns FALSE, your trap handler should execute appropriate customized error recovery.

Output

The Boolean value of FILTER is output as the bottom bit of the AL register. The input word ERRORS287 is popped from the 80286 stack upon return.

If FILTER returns FALSE, the 80287 is returned to the same state that it was prior to the call to FILTER.

If FILTER returns TRUE, the 80287 is changed to reflect a retry of the instruction with normal operands (in case of a D error) or I masked (in case of an I error).

PL/M-286 Programming Example

```
FILTER_MODULE: DO;

FILTER: PROCEDURE (ERRORS287) BYTE EXTERNAL;
  DECLARE ERRORS287 WORD;
END;

/* SIMPLE_TRAP_HANDLER is a floating-point exception
   handler which provides a bare minimum interface to
   EH287.LIB. If the error which caused this interrupt has
   been handled by FILTER, then the interrupt returns to
   the user program immediately. Otherwise, your inserted
   custom code is executed. */

SIMPLE_TRAP_HANDLER: PROCEDURE INTERRUPT PUBLIC;

  DECLARE ERRORS287 WORD;

  IF FILTER (ERRORS287 := GET$REAL$ERROR) THEN RETURN;
```

```
   /* Here you could place code to handle exceptions other
      than the denormals and nontrapping NaNs handled by
      FILTER. */

END SIMPLE_TRAP_HANDLER;

END FILTER_MODULE;
```

ASM286 Programming Example

```
NAME FILTER_MODULE

EXTRN FILTER: FAR

CODE SEGMENT ER PUBLIC

STACK_LAYOUT STRUC              ; Pointed at by BP during
                                ; SIMPLE_TRAP_HANDLER

; You can place additional stack-allocated variables here,
; for your custom code. You refer to them as
; BP.your_var_name.

ERRORS287       DW ?            ; 80287 status word before
                                ; clearing
REGISTERS_286   DW 8 DUP(?)     ; Pushed by PUSH_REGISTERS
FLAGS_286       DW ?            ; Pushed by 80286 when interrupt
                                ; called
RET_ADDRESS     DD ?            ; From SIMPLE_TRAP_HANDLER

STACK_LAYOUT ENDS

; SIMPLE_TRAP_HANDLER is a floating-point exception
; handler which provides a bare minimum interface to
; EH287.LIB. If the error which caused this interrupt has
; been handled by FILTER, then the interrupt returns to
; the user program immediately. Otherwise, your inserted
; custom code is executed.

SIMPLE_TRAP_HANDLER PROC FAR

 CALL PUSH_REGISTERS            ; Save the 80286 registers
 SUB SP,OFFSET REGISTERS_286;   Allocate room for stack
                                ; layout
 MOV BP,SP                      ; Set up indexing into
                                ; STACK_LAYOUT
 FNSTSW [BP].ERRORS287          ; Save the errors that caused
                                ; the exception
 MOV AX,[BP].ERRORS287          ; Fetch the status word ...
 PUSH AX                        ; ... and store it as FILTER
                                ; parameter
 FNCLEX                         ; Clear the exceptions so
                                ; handler can use 80287
 CALL FILTER                    ; Check for denormal or non-
                                ; trapping NaN
 RCR AL,1                       ; Was error handled?
 JC TRAP_EXIT                   ; If so then do nothing more
```

```
  ; Here you could place code to handle exceptions other
  ; than the denormals and nontrapping NaNs handled by
  ; FILTER.

TRAP_EXIT:
   CALL POP_REGISTERS          ; Restore the 80286 registers
   IRET

SIMPLE_TRAP_HANDLER ENDP

; PUSH_REGISTERS causes the eight 80286 registers
; SI,DI,ES,BP,DX,CX,BX,AX to be pushed onto the 80286
; stack. The registers can be popped off with a call to
; POP_REGISTERS.

PUSH_REGISTERS PROC NEAR
 PUSH DI
 PUSH ES
 PUSH BP
 PUSH DX
 PUSH CX
 PUSH BX
 PUSH AX
 MOV BP,SP           ; Get stack pointer into an index
                     ; register
 XCHG SI,[BP+14]     ; Exchange the return address with SI
                     ; which is being saved
 JMP SI              ; This is the RETURN of the procedure
PUSH_REGISTERS ENDP

; POP_REGISTERS causes the eight registers which were
; pushed by PUSH_REGISTERS to be popped, restoring them to
; their original values.

POP_REGISTERS PROC NEAR
 POP SI              ; Hold this call's return address
                     ; temporarily in SI
 MOV BP,SP           ; Get the stack pointer into an index
                     ; register
 XCHG SI,[BP+14]     ; Restore SI, and position the return
                     ; address
 POP AX
 POP BX
 POP CX
 POP DX
 POP BP
 POP ES
 POP DI
 RET          ; Return address is in position due to above
              ; XCHG
POP_REGISTERS ENDP

CODE ENDS

END
```

NORMAL — Detect and normalize D error nonnormal arguments

Input

NORMAL expects the 80287 to be completely cleared to its initialized state. Information about the 80287 is obtained from ESTATE287, not from the entry state of the 80287 itself.

Two parameters, totalling six bytes, must be pushed onto the 80286 stack before calling NORMAL.

First, the four-byte pointer ESTATE287_PTR is pushed. The segment of ESTATE287_PTR is pushed first, followed by the offset. ESTATE287_PTR points to the 144-byte ESTATE287 structure that was filled by a previous call to DECODE.

Second, a two-byte quantity, the bottom byte of which is ERRORS287, is pushed onto the stack. ERRORS287 is the bottom byte of the 80287 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of the two-byte quantity is ignored.

Function

NORMAL first checks ERRORS287 to see if the D error bit is the only unmasked error bit that is set. If this is not the case, NORMAL immediately returns FALSE, indicating that no normalization retry is to take place.

If D is set and no other unmasked error bits are set, NORMAL returns TRUE. If the operation that caused the D error was a load operation, the nonnormal arguments are left unchanged. If the operation was not a load operation, NORMAL modifies the denormal arguments. Denormal SHORT_REAL and LONG_REAL arguments are normalized; denormal TEMP_REAL arguments are converted to zero. Only the copies of arguments in ESTATE287's ARG1 and ARG2 fields are modified.

Whenever NORMAL returns TRUE, you should call ENCODE with RETRY _FLAG set to TRUE. You may leave D unmasked, using the same control word in effect when the trap handler was called. If the operation was a load instruction, ENCODE will perform the load with D masked and not cause a repeat of the D error.

Note that NORMAL always returns FALSE if the operation that caused the trap is DC287 or a CEL287 function. Only individual 80287 instructions can cause a D error.

Output

NORMAL returns normalized arguments by modifying ESTATE287. In addition, it returns a Boolean value in the AL register, indicating further action to be taken. If the bottom bit of AL is 1, a normalization has taken place, and NORMAL is requesting the caller to retry the offending operation by calling ENCODE with a true RETRY_FLAG. If the bottom bit of AL is 0, no such retry is requested.

NORMAL returns the 80287 itself to the same cleared state it had upon entry.

PL/M-286 Declaration and Calling Sequence

```
NORMAL: PROCEDURE (ESTATE287_PTR,ERRORS287) BYTE EXTERNAL;
  DECLARE ESTATE287_PTR POINTER, ERRORS287 WORD;
END;
```

```
/* Assume ESTATE287 is already declared as presented
   earlier in this chapter */

DECLARE ERRORS287 WORD;
DECLARE NORM_RETRY BYTE;

NORM_RETRY = NORMAL( @ESTATE287, ERRORS287 );

/* Now NORM_RETRY is true if a retry of the operation
   should be made */
```

ASM286 Declaration and Calling Sequence

```
; the following line must occur outside of all SEGMENT-
; ENDS pairs

    EXTRN NORMAL: FAR

DATA SEGMENT
    ESTATE_287        ESTATE_TEMPLATE
    ERRORS287         DW  ?
    NORM_RETRY        DB  ?
DATA ENDS

; the following code assumes that DS contains the segment
; of ESTATE287

 ASSUME DS:DATA
 PUSH DS                ; push the segment of ESTATE287 onto
                        ; the stack
MOV AX,OFFSET(ESTATE287)
PUSH AX                 ; 4-byte pointer is now pushed onto
                        ; stack
 MOV AL,ERRORS287       ; second parameter
 PUSH AX                ; pushed onto stack
 CALL NORMAL            ; AL now tells whether to retry
 MOV NORM_RETRY,AL      ; Boolean answer stored in NORM_RETRY
```

SIEVE — Detects nontrapping NaNs that should be ignored

Input

SIEVE expects the 80287 to be completely cleared to its initialized state. Information about the 80287 is obtained from ESTATE287, not from the entry state of the 80287 itself.

Two parameters, totalling six bytes, must be pushed onto the 80286 stack before calling SIEVE.

First, the four-byte pointer ESTATE287_PTR is pushed. The segment of ESTATE287_PTR is pushed first, followed by the offset. ESTATE287_PTR points to the 144-byte ESTATE287 structure that was filled by a previous call to DECODE.

Second, a two-byte quantity, the bottom byte of which is ERRORS287, is pushed onto the stack. ERRORS287 is the bottom byte of the 80287 status word as it existed when the trap handler was called, before the exceptions were cleared. The top byte of the two-byte quantity is ignored.

Function

SIEVE signals those cases in which the I exception should not have been given because the argument was a legal nontrapping NaN. This detection applies to all arithmetic operations that check for NaN inputs.

SIEVE follows the conventions described in "Nontrapping NaNs" and "Nonordered Comparisons" near the beginning of this chapter. If it is determined that this I error should not have taken place because the NaN arguments are nontrapping and the operation is not an ordered comparison, SIEVE returns TRUE. In this case, the caller should retry the offending operation by calling ENCODE with a true RETRY _FLAG and CONTROL287 modified so that I is masked. ENCODE restores the original control word, with I unmasked, after the retry is executed.

If it is determined that this I error should still be flagged (for example, a stack error has occurred), SIEVE returns FALSE. Some appropriate I error recovery action should be performed.

Note that if the operation that caused the trap is DC287 or a CEL287 function, SIEVE always returns FALSE. All legal nontrapping NaNs arise from individual 80287 instructions.

Output

The Boolean answer is returned as the bottom bit of the AL register. If this bit is 1, the I error should not have taken place, (according to the nontrapping NaN conventions described earlier in this chapter).

SIEVE leaves the 80287 in the same cleared state it had upon entry. The ESTATE287 structure is also unchanged by SIEVE.

PL/M-286 Declaration and Calling Sequence

```
SIEVE: PROCEDURE (ESTATE287_PTR,ERRORS287) BYTE EXTERNAL;
  DECLARE ESTATE287_PTR POINTER, ERRORS287 WORD;
END;
```

```
/* Assume ESTATE287 is already declared as presented
   earlier in this chapter */

DECLARE ERRORS287 WORD;
DECLARE LEGAL_NAN BYTE;

LEGAL_NAN = SIEVE( @ESTATE287, ERRORS287 );

/* Now LEGAL_NAN is true if there should not have been an
   "I" error */
```

ASM286 Declaration and Calling Sequence

```
; the following line must occur outside of all SEGMENT-
; ENDS pairs

     EXTRN   SIEVE: FAR

DATA SEGMENT
     ESTATE_287   ESTATE_TEMPLATE
     ERRORS287    DW   ?
DATA ENDS

; the following code assumes that DS contains the segment
; of ESTATE287

     ASSUME DS:DATA
     PUSH DS                ; push the segment of ESTATE287 onto
                            ; the stack
     MOV AX,OFFSET(ESTATE287)
     PUSH AX                ; 4-byte pointer is now pushed onto
                            ; stack
     MOV AL,ERRORS287       ; second parameter
     PUSH AX                ; pushed onto stack
     CALL SIEVE             ; AL now contains Boolean answer
     MOV LEGAL_NAN,AL       ; LEGAL_NAN true if there is no
                            ; "I" error
```

Binding EH287.LIB to Program Modules

This section tells how to bind EH287.LIB into your program using a BND286 command.

If you are linking both CEL287.LIB and EH287.LIB, it is absolutely necessary that EH287.LIB appear after CEL287.LIB in the BND286 command for this reason: EH287.LIB contains references to all the functions of CEL287.LIB. However, if there are any unused CEL287 functions, they do not need to be linked for EH287.LIB to work correctly. We have therefore provided public symbols for CEL287.LIB in EH287.LIB, all of which point to an 80286 HLT instruction. CEL287.LIB must appear first in the BND286 invocation to pick up the actual references to CEL287.LIB functions; then EH287.LIB will supply HLT references (which do not add to the code size) for all unused CEL287 symbols.

If you mistakenly put CEL287.LIB after EH287.LIB in your bind invocation, the bind will perform without any visible problems, but all calls to CEL287.LIB will go to a HLT instruction. When you attempt to execute your program, the NDP will halt the first time a CEL287 function is called.

You must bind in the 80287.LIB when using EH287.LIB.

Following is the recommended module order in the BND286 statement:

Your object modules
DC287.LIB (if your program uses it)
CEL287.LIB (if your program uses it)
EH287.LIB
80287.LIB

For example, if your modules are MYMOD1.OBJ and MYMOD2.OBJ, issue the following command:

```
-BND286 :F1:MYMOD1.OBJ, &<cr>
        :F1:MYMOD2.OBJ, &<cr>
        :F0:DC287.LIB, &<cr>
        :F0:CEL287.LIB, &<cr>
        :F0:EH287.LIB, &<cr>
        :F0:80287.LIB &<cr>
        OBJECT (:F1:MYPROG.LNK)
```

If you have a single module MYPROG.OBJ and do not use any libraries other than EH287.LIB, issue the following command:

```
-BND286 :F1:MYPROG.OBJ, &<cr>
        :F0:EH287.LIB, &<cr>
        :F0:80287.LIB &<cr>
        OBJECT (:F1:MYPROG.LNK)
```

NOTE

If you are writing your own specialized interrupt handler, you will need to use BLD286 to create the interrupt table entry. Also, you must include the NOLOAD control in the above BND286 command lines.

CHAPTER 6
IMPLEMENTING THE IEEE STANDARD

The NDP, together with the 80287 Support Library, provides an implementation of "A Proposed Standard for Binary Floating-Point Arithmetic," Draft 10.0 Task P754, December 2, 1982.

This chapter describes the relationship between the NDP and the IEEE Standard. It gives the choices made in this book for the places where the Standard has options. It pinpoints two areas in which the book does not conform to the Standard. If your application requires it, you can provide software to meet the Standard in these areas. The book indicates how to write this software.

This chapter contains many terms that have precise technical meanings, as specified by the Standard. The Standard's convention of emphasizing the precise meaning by capitalizing those terms is used. The Glossary provides the definitions for all capitalized phrases in this chapter.

Options Chosen

The SHORT_REAL and LONG_REAL formats conform perfectly to the Standard's Single and Double Floating-Point Numbers, respectively. The TEMP_REAL format is the same as the Standard's Double Extended format. The Standard allows a choice of Bias in representing the exponent; the Bias 16383 decimal has been chosen.

For the Double Extended format, the Standard contains an option for the meaning of the minimum exponent combined with a nonzero significand. The Bias for this special case can be either 16383, as in all the other cases, or 16382, making the smallest exponent equivalent to the second-smallest exponent. The Bias 16382 has been chosen for this case, allowing you to distinguish between Denormal numbers (integer part is zero, fraction is nonzero, Biased Exponent is 0) and Unnormal numbers of the same value (same as the denormal except the Biased Exponent is 1.)

The Standard endorses the support of only one extended format. That format is provided here.

The Standard allows you to specify which NaNs are Trapping and which are Nontrapping. EH287.LIB, which provides a software implementation of Nontrapping NaNs, defines the distinction. If the most significant bit of the fractional part of a NaN is 1, the NaN is Nontrapping. If it is 0, the NaN is Trapping.

When a masked I error involves two NaN inputs, the Standard lets you set the rule for which NaN is output. The NaN whose absolute value is greatest has been chosen.

Areas Needing the Support Library to Meet the Standard

The standard has five features that are not implemented directly on the 80287.

1. The Standard requires you to provide a Normalizing Mode in which any nonnormal operands to functions are automatically normalized before the function is performed. The NPX instead has a D exception (not mentioned in the Standard), which gives the exception handler the opportunity to perform the normalization specified by the Standard. The D exception handler provided by EH287.LIB completely implements the Standard's Normalizing Mode for Single and Double

precision arguments. Normalizing mode for Double Extended operands is implemented with one non-Standard feature, mentioned in the next section.

2. The Standard specifies the "equal" comparison test yield an answer of FALSE, with no I error if the relation between the input operands is unordered. The 80287 FCOM and FTST instructions issue an I error for this case. The EH87.LIB error handler filters out the I error, using the following convention: whenever an FCOM or FTST instruction is followed by a MOV AX,AX instruction (8BC0 hex), and neither argument is a trapping NaN, the error handler assumes that a Standard equal comparison was intended and returns the correct answer with I erased. Note that I must be unmasked for this action to occur.
3. The Standard requires you to provide two kinds of NaNs, Trapping and Nontrapping. Nontrapping NaNs do not cause further I errors when they occur as operands to calculations. The NPX directly supports only Trapping NaNs. The EH287.LIB error handler module implements Nontrapping NaNs by returning the correct answer with I erased. Note that I must be unmasked for this action to occur.
4. The Standard requires all functions that convert real numbers to integer formats to automatically normalize the inputs if necessary. The 80287 FIST instruction does not do so. CEL287.LIB's integer conversion functions fully meet the Standard in this aspect.
5. The Standard specifies the remainder function that is provided by mqerRMD in CEL287.LIB. The 80287 FPREM instruction returns answers in a different range.

Further Software Required to Meet the Standard

Two cases exist in which you will need to provide further software to meet the Standard. This software is not provided, because a vast majority of applications never encounter these cases.

1. Nontrapping NaNs are not implemented when I is masked. Likewise, the Standard's "equal" function is not implemented when I is masked. You can simulate the Standard's concept of a masked I exception by unmasking the 80287 I bit and providing an I trap handler that supports Nontrapping NaNs and the "equal" function, but otherwise acts just as if I were masked. Chapter 5 contains examples in both ASM286 and PL/M-286 for doing this.
2. Denormal operands in the TEMP_REAL format are converted to 0 by the EH287.LIB Normalizing Mode, giving sharp Underflow to 0. The Standard specifies that the operation be performed on the real numbers represented by the Denormals, giving gradual Underflow. To correctly perform such arithmetic, you will have to normalize the operands into a format identical to TEMP_REAL, except for two extra exponent bits, then perform the operation on those numbers. Thus, your software must manage the 17-bit exponent explicitly.

It is probably disadvantageous for most users to increase the size of the Normalizing routine by the amount necessary to provide this expanded arithmetic. The TEMP_REAL exponent field is much larger than the LONG_REAL exponent field; thus, it is extremely unlikely that you will encounter TEMP_REAL Underflow. If meeting the Standard is a more important criterion to you than the choice between Normalizing and warning modes, you can select warning mode (D masked), which fully meets the Standard.

If you do wish to implement the TEMP_REAL arithmetic with extra exponent bits, see the following useful pointers about when the D error occurs:

a. TEMP_REAL numbers are considered Denormal by the NPX whenever the Biased Exponent is 0 (minimum exponent). This is true even if the explicit integer bit of the significand is 1. Such numbers can occur as the result of Underflow.

b. The 80287 FLD instruction can cause a D error if a number is being loaded from memory. It cannot cause a D error if it is coming from elsewhere in the 80287 stack.

c. The 80287 FCOM and FTST instructions cause a D error for unnormal operands as well as denormal operands.

d. When both D and I errors occur, you will want to know which is signalled first. When a comparison instruction is issued between a nonexistent stack element and a Denormal number in 80286 memory, the D and I errors are issued simultaneously. In all other situations, a stack I error takes precedence over a D error, and a D error takes precedence over a nonstack I error.

The 80287 Support Library provides internal support for Pascal-286 programs that require manipulation of Real data types and error handling capabilities. The appropriate libraries must be bound to the Pascal object code to ensure complete numeric function.

CEL287.LIB

Pascal programs that use any of the following functions must be combined with the CEL287 Library to prevent any unresolved external errors from occuring. If these functions are not used, the CEL287 Library is not required. The functions that require CEL287 support are SQR, SQRT, EXP, LN, SIN, COS, TAN, ARCSIN, ARCCOS, ARCTAN, TRUNC, ROUND, LTRUNC, and LROUND.

EH287.LIB

To implement the IEEE Math Standard and provide error handling capabilities, the EH287 Library must be bound to the Pascal object code. The IEEE standard is provided through Rounding, Precision, and Normalization controls in the Library. Two Pascal functions also use the Library for error handling—GET8087ERRORS and MASK8087ERRORS. Because of the controls provided by this library, any Pascal-286 program that performs numeric manipulations should be bound to this library. The formats of the two above functions are as follows:

`GET8087ERRORS (`*variable*`)`

where *variable* is of the type AT87ERRORS:

```
Type AT87ERRORS = SET OF AT87EXCEPTIONS;
     AT87EXCEPTIONS =
          (AT87NVLD, AT87DENR, AT87ZDIX, AT87OVER,
                AT87UNDR, AT87PRCN, AT87RSVD);
```

Note that:

- AT87NVLD, AT87DENR, AT87ZDIV, AT87OVER, AT87UNDR, and AT87PRCN represent 80287 exception (or error) flags.
- AT87RSVD is the Reserved location and AT87MASK is the Interrupt enable bit of the 80287.
- GET8087ERRORS returns the values of the flag bits for use with error handling routines.

Example

`GET8087ERRORS (`*errors*`);`

where

errors is of type AT87ERRORS, defined above.

`MASK8087ERRORS (`*expression*`)`

where

expression indicates which 80287 exceptions will be masked or disabled.

This function sets the Real error mask on the 80287 and can be used to change the numeric controls used in processing Real arithmetic.

Examples

```
MASK8087ERRORS ([AT87DENR, AT87ZDN, AT87OVER,
     AT87UNDR, AT87PRCN]),
```

This is the initial setting. It corresponds to masking all exceptions except INVALID and enabling Interrupts.

```
MASK8087ERRORS ([]);
```

This resets the error mask such that all errors are unmasked and all interrupts are enabled.

APPENDIX B
THE 80287 EMULATOR

The 80287 emulator is a software module that exactly duplicates all the functions of the 80287. You can use the emulator to develop prototypes for systems that will have an 80287 or you can use the emulator if you need the 80287's floating-point capabilities but do not need its speed.

The 80287 emulator is available in the *8086 Software Toolbox*. Instructions for using the emulator are contained in the *8086 Software Toolbox Manual*, order number 122203.

For more information on this product, contact your Intel representative.

APPENDIX C SUMMARY OF 80287 FLOATING-POINT FORMATS

The 80287 supports three real number formats. All formats are in reverse order when stored in 80286 memory. That is, the first byte (lowest memory address) is the least significant byte of the Significand; the last byte (highest memory address) contains the sign and the top seven bits of the Exponent.

The three formats supported are the 32-bit Short Real, the 64-bit Long Real, and the 80-bit Temporary Real.

Short Real

1-bit Sign	
8-bit Exponent:	Bias 126 if Exponent zero; 127 if nonzero
0-bit Implicit Integer Bit:	0 if Exponent zero; 1 if nonzero
23-bit Fractional Part:	digits to the right of the Binary Point

Long Real

1-bit Sign	
11-bit Exponent:	Bias 1022 if Exponent zero; 1023 if nonzero
0-bit Implicit Integer Bit:	0 if Exponent zero; 1 if non-zero
52-bit Fractional Part:	digits to the right of the Binary Point

Temporary Real

1-bit Sign	
15-bit Exponent:	Bias 16382 if Exponent zero; 16383 if nonzero
1-bit Explicit Integer Bit	
63-bit Fractional Part:	digits to the right of the Binary Point

Special Values in All Three Formats

Infinity:	maximum exponent and zero Significand
NaN:	maximum exponent and nonzero Significand
Zero:	minimum exponent and zero Significand
Denormal:	minimum exponent and nonzero Significand

APPENDIX D
SUMMARY OF 80287 INSTRUCTIONS

This appendix contains a reference chart of NPX instructions. Following is an explanation of each column of the chart.

Opcode

This column gives the machine codes generated by ASM86 and BND286 for the given instruction. Digits and uppercase letters A–F are hexadecimal digits. In addition, the following special codes are used:

- i denotes a 4-bit quantity the top bit of which is 0 and the bottom three bits of which give the 80287 stack element number for the instruction.
- j denotes i plus 8. It is a 4-bit quantity the top bit of which is 1 and the bottom three bits of which give the 80287 stack element number for the instruction.
- / followed by a digit denotes a MODRM byte, as described in the ASM286 language manual. The digit gives the value of the middle REG field of the byte (bits 5,4,3). The MOD field (bits 7,6) can be any of the values 00, 01, or 10. The R/M field (bits 2,1,0) can be any of the 8 possible values. For some values of MOD and R/M, one or two immediate displacement bytes follow the MODRM byte, as defined by the 80286 architecture.

The machine codes are those for the 80287 component, produced by binding 80287.LIB to your ASM286 object modules. If a segment override byte exists, it goes between the first (WAIT or NOP) byte and the second (ESCAPE) byte.

Instruction

This column gives the 80287 instruction just as it appears in an ASM286 program. Operands that begin with "mem" can be replaced with any memory operand denoting a data area of the correct number of bytes. The number following mem gives the decimal number of memory bytes acted upon. A trailing r in the name denotes a REAL format, i an INTEGER format, and d a Binary Coded Decimal format.

Function

This column gives a concise description of the function performed by the instruction.

Clocks

This column gives the typical number of clock cycles used by the 80287 chip to execute the instruction. It is not an exact number. If a MODRM byte is involved, a typical time of nine cycles is added for calculating the effective address of the memory operand.

ErrID

This column gives the hexadecimal value returned by the procedure DECODE in the library EH287.LIB, described in Chapter 5. The value indicates the type of instruction that caused an error, and is returned in the OPERATION field of the structure ESTATE287.

Errors

This column lists the possible exceptions that can occur if the instruction is executed.

Opcode	Instruction	Function	Clocks	ErrID	Errors
9B D9 F0	F2XM1	ST←(2**ST)−1	500	19	UP
9B D9 E1	FABS	ST←I ST I	14	01	I
9B DE C1	FADD	ST(1)←ST(1)+ST,pop	90	05	IDOUP
9B DC Ci	FADD ST(i),ST	ST(i)←ST(i)+ST	85	05	IDOUP
9B D8 Ci	FADD ST,ST(i)	ST←(i)+ST	85	05	IDOUP
9B D8 /0	FADD mem4r	ST←ST+mem4r	114	05	IDOUP
9B DC /0	FADD mem8r	ST←ST+mem8r	119	05	IDOUP
9B DE Ci	FADDP ST(i),ST	ST(i)←ST(i)+ST,pop	90	05	IDOUP
9B DF /4	FBLC mem10d	push, ST←mem10d	309	10	I
9B DF /6	FBSTP mem10d	mem10d←ST,pop	539	0F	I
9B D9 E0	FCHS	ST←−ST	15	02	I
9B DB E2	FCLEX	clear exceptions	5		
9B D8 D1	FCOM	compare ST−ST(1)	45	03	ID
9B D8 Di	FCOM ST(i)	compare ST−ST(i)	45	03	ID
9B D8 /2	FCOM mem4r	compare St−mem4R	74	03	ID
9B DC /2	FCOM mem8r	compare ST−mem8r	79	03	ID
9B D8 D9	FCOMP	compare ST−ST(1),pop	47	03	ID
9B D8 Dj	FCOMP ST(i)	compare ST−ST(i),pop	47	03	ID
9B D8 /3	FCOMP mem4r	compare ST−mem4r,pop	77	03	ID
9B DC /3	FCOMP mem8r	compare St−mem8r,pop	81	03	ID
9B DE D9	FCOMPP	compare ST−ST(1),pop2	50	03	ID
9B D9 F6	FDECSTP	decrement stack pointer	9		
9B DE F9	FDIV	ST(1)←ST(1)/ST,pop	202	09	IDZOUP
9B DC Fj	FDIV ST(i),ST	ST(i)←ST(i)/ST	198	09	IDZOUP
9B D8 Fi	FDIV ST,ST(i)	ST←ST/ST(i)	198	09	IDZOUP
9B D8 /6	FDIV mem4r	ST←ST/mem4r	229	09	IDZOUP
9B DC /6	FDIV mem8r	ST←ST/mem8r	234	09	IDZOUP
9B DE Fi	FDIVP ST(i),ST	ST(i)←ST(i)/ST,pop	202	09	IDZOUP
9B DE F1	FDIVR	ST(1)←ST/ST(1),pop	203	0A	IDZOUP
9B DC Fj	FDIVR ST(i),ST	ST(i)←ST/ST(i)	199	0A	IDZOUP
9B D8 Fj	FDIVR ST,ST(i)	ST←ST(i)/ST	199	0A	IDZOUP
9B D8 /7	FDIVR mem4r	ST←mem4r/ST	230	0A	IDZOUP
9B DC /7	FDIVR mem8r	ST←mem8r/ST	235	0A	IDZOUP
9B DE Fi	FDIVRP ST(i),ST	ST(i)←ST/ST(i),pop	203	0A	IDZOUP
9B DD Ci	FFREE ST(i)	empty ST(i)	11		
9B DE /0	FIADD mem2i	ST←ST+mem2i	129	05	IDOP
9B DA /0	FIADD mem4i	ST←ST+mem4i	134	05	IDOP
9B DE /2	FICOM mem2i	compare ST−mem2i	89	03	ID
9B DA /2	FICOM mem4i	compare ST−mem4i	94	03	ID
9B DE /3	FICOMP mem2i	compare ST−mem2i,pop	91	03	ID
9B DA /3	FICOMP mem4i	compare ST−mem4i,pop	96	03	ID
9B DE /6	FIDIV mem2i	ST←ST/mem2i	239	09	IDZOUP
9B DA /6	FIDIV mem4i	ST←ST/mem4i	245	09	IDZOUP
9B DE /7	FIDIVR mem2i	ST←mem2i/ST	239	0A	IDZOUP
9B DA /7	FIDIVR mem4i	ST←mem4i/ST	246	0A	IDZOUP
9B DF /0	FILD mem2i	push,ST←mem2i	59	10	I
9B DB /0	FILD mem4i	push,ST←mem4i	65	10	I
9B DF /5	FILD mem8i	push,ST←mem8i	73	10	I
9B DE /1	FIMUL mem2i	ST←ST*mem2i	139	08	IDOP
9B DA /1	FIMUL mem4i	ST←ST*mem4i	145	08	IDOP
9B D9 F7	FINCSTP	increment stack pointer	9		
9B DB E3	FINIT	initialize 80287	5		
9B DF /2	FIST mem2i	mem2i←ST	95	0F	IP
9B DB /2	FIST mem4i	mem4i←ST	97	0F	IP
9B DF /3	FISTP mem2i	mem2i←ST,pop	97	0F	IP
9B DB /3	FISTP mem4i	mem4i←ST,pop	99	0F	IP
9B DF /7	FISTP mem8i	mem8i←ST,pop	109	0F	IP
9B DE /4	FISUB mem2i	ST←ST−mem2i	129	06	IDOP
9B DA /4	FISUB mem4i	ST←ST−mem4i	134	06	IDOP
9B DE /5	FISUBR mem2i	ST←mem2i−ST	129	07	IDOP
9B DA /5	FISUBR mem4i	ST←mem4i−ST	134	07	IDOP
9B D9 Ci	FLD ST(i)	push,ST←old ST(i)	20	10	I
9B DB /5	FLD mem10r	push,ST←mem10r	66	10	ID
9B D9 /0	FLD mem4i	push,ST←mem4r	52	10	ID
9B DD /0	FLD mem8r	push,ST←mem8r	55	10	ID

Opcode			Instruction	Function	Clocks	ErrID	Errors
9B	D9	E8	FLD1	push,ST ← 1.0	18	11	I
9B	D9	/5	FLDCW mem2i	control word ← mem2i	19		
9B	D9	/4	FLDENV mem14	environment ← mem14	49		
9B	D9	EA	FLDL2E	push,ST ← log base 2 of e	18	13	I
9B	D9	E9	FLDL2T	push,ST ← log base 2 of 10	19	12	I
9B	D9	EC	FLDLG2	push,ST ← log bse 10 of 2	21	15	I
9B	D9	ED	FLDLN2	push,ST ← log base e of 2	20	16	I
9B	D9	EB	FLDPI	push,ST ← Pi	19	14	I
9B	D9	EE	FLDZ	push,ST ← +0.0	14	17	I
9B	DE	C9	FMUL	ST(1) ← ST(1)*ST,pop	142	08	IDOUP
9B	DC	Cj	FMUL ST(i),ST	ST(I) ← ST(i)*ST	138	08	IDOUP
9B	D8	Cj	FMUL ST,ST(i)	ST ← ST*ST(i)	138	08	IDOUP
9B	D8	/1	FMUL mem4r	ST ← ST*mem4r	127	08	IDOUP
9B	DC	/1	FMUL mem8r	ST ← ST*mem8r	170	08	IDOUP
9B	DE	Cj	FMULP ST(i),ST	ST(i) ← ST(i)*ST,pop	142	08	IDOUP
90	DB	E2	FNCLEX	nowait clear exceptions	5		
90	DB	E3	FNINIT	nowait initialize 80287	5		
9B	D9	D0	FNOP	no operation	13		
90	DD	/6	FNSAVE mem94	nowait mem94 ← 80287 state	219		
90	D9	/7	FNSTCW mem2i	nowait mem2i ← control word	24		
90	D9	/6	FNSTENV mem14	nowait mem14 ← environment	54		
90	DD	/7	FNSTSW mem2i	nowait mem2i ← status word	24		
9B	D9	F3	FPATAN	ST ← arctan(ST(1)/ST),pop	650	1D	UP
9B	D9	F8	FPREM	ST ← REPEAT(ST − ST(1))	125	1E	IDU
9B	D9	F2	FPTAN	push,ST(1)/ST ← tan(old ST)	400	1C	IP
9B	D9	FC	FRNDINT	ST ← round(ST)	45	1F	IP
9B	DD	/4	FRSTOR mem94	80287 state ← mem94	219		
9B	DD	/6	FSAVE mem94	mem94 ← 80287 state	219		
9B	D9	FD	FSCALE	ST ← ST*2**ST(1)	35	18	IOU
9B	D9	FA	FSQRT	ST ← square root of ST	183	0C	IDP
9B	DD	Di	FST ST(i)	ST(i) ← ST	18	0F	I
9B	D9	/2	FST mem4r	mem4r ← ST	96	0F	IOUP
9B	DD	/2	FST mem8r	mem8r ← ST	109	0F	IOUP
9B	D9	/7	FSTCW mem2i	mem2i ← control word	24		
9B	D9	/6	FSTENV mem14	mem14 ← environment	54		
9B	DD	Dj	FSTP ST(i)	ST(i) ← ST,pop	20	0F	I
9B	DB	/7	FSTP mem10r	mem10r ← ST,pop	64	0F	I
9B	D9	/3	FSTP mem4r	mem4r ← ST,pop	98	0F	IOUP
9B	DD	/3	FSTP mem8r	mem8r ← ST,pop	111	0F	IOUP
9B	DD	/7	FSTSW mem2i	mem2i ← status word	24		
9B	DE	E9	FSUB	ST(1) ← ST(1) − ST,pop	90	06	IDOUP
9B	DC	Ej	FSUB ST(i),ST	ST(i) ← ST(i) − ST	85	06	IDOUP
9B	D8	Ei	FSUB ST,ST(i)	ST ← ST − ST(i)	85	06	IDOUP
9B	D8	/4	FSUB mem4r	ST ← ST − mem4r	114	06	IDOUP
9B	DC	/4	FSUB mem8r	ST ← ST − mem8r	119	06	IDOUP
9B	DE	Ej	FSUBP ST(i),ST	ST(i) ← ST(i) − ST,pop	90	06	IDOUP
9B	DE	E1	FSUBR	ST(1) ← ST − ST(1),pop	90	07	IDOUP
9B	DC	Ei	FUSBR ST(i),ST	ST(i) ← ST − ST(i)	87	07	IDOUP
9B	D8	Ej	FSUBR ST,ST(i)	ST ← ST(i) − ST	87	07	IDOUP
9B	D8	/5	FSUBR mem4r	ST ← mem4r − ST	114	07	IDOUP
9B	DC	/5	FSUBR mem8r	ST ← mem8r − ST	119	07	IDOUP
9B	DE	Ei	FSUBRP ST(i),ST	ST(i) ← ST − ST(i),pop	90	07	IDOUP
9B	D9	E4	FTST	compare ST − 0.0	42	04	ID
9B			FWAIT	wait for 80287 ready			
9B	D9	E5	FXAM	C3—C0 ← type of ST	17		
9B	D9	C9	FXCH	exchange ST and ST(1)	12	0E	I
9B	D9	Cj	FXCH ST(i)	exchange ST and ST(i)	12	0E	I
9B	D9	F4	FXTRACT	push, ST(1) ← expo, ST ← sig	50	0B	I
9B	D9	F1	FYL2X	ST ← ST(1l)*log2(ST),pop	950	1A	P
9B	D9	F9	FYL2XP1	ST ← ST(1)*log2(ST + 1),pop	850	1B	P

APPENDIX E SUMMARY OF SUPPORT LIBRARY FUNCTIONS

This appendix gives condensed summaries of the procedures and functions of the DC287.LIB, CEL287.LIB, and EH287.LIB libraries. Use this appendix as a quick reference after assimilating the material in Chapters 2, 3, 4, and 5 of the manual.

DC287.LIB

All input parameters to DC287.LIB are 4-byte long pointers to 80286 memory buffers that contain the inputs and outputs. The pointer(s) are pushed onto the 80286 stack, high byte first, before calling the procedure. The procedure returns with the pointer(s) popped off the 80286 stack.

The first three procedures have one input pointer. The last four procedures have two input pointers.

mqcBIN_DECLOW (Convert binary number to decimal string)

Input: BIN_DECLOW_BLOCK_PTR →
4-byte BIN_PTR → input binary number
1-byte BIN_TYPE: 0 for SHORT_REAL
1 for LONG_REAL
2 for TEMP_REAL
1-byte DEC_LENGTH: length of output field
4-byte DEC_PTR → output decimal significand, decimal point at right
2-byte DEC_EXPONENT: base ten exponent of output
1-byte DEC_SIGN: sign of output, in ASCII

Sign and digits output for unusual inputs:
NaN = ".."
+INFINITY = "++"
−INFINITY = "−−"
+0 = "0 "
−0 = "−0"

Errors: I,D,P

mpqDEC_BIN (Convert decimal string to binary number)

Input: DEC_BIN_BLOCK_PTR →
4-byte BIN_PTR → output binary number
1-byte BIN_TYPE: 0 for SHORT_REAL
1 for LONG_REAL
2 for TEMP_REAL
1-byte DEC_LENGTH: length of input field
4-byte DEC_PTR → input string.

Errors: O,U,P

mqcDECLOW_BIN (Convert decimal string, low-level interface, to binary number)

Input: DECCLOW_BIN_BLOCK_PTR →
4-byte BIN_PTR → output binary number
1-byte BIN_TYPE: 0 for SHORT_REAL
1 for LONG_REAL
2 for TEMP_REAL
1-byte DEC_LENGTH: length of input field
4-byte DEC_PTR → input string, stripped down to decimal digits
2-byte DEC_EXPONENT: base ten exponent, with decimal point to right of input
1-byte DEC_SIGN: sign of input, in ASCII

Errors: O,U,P

mqcLONG_TEMP (Convert LONG_REAL to TEMP_REAL)

Inputs: LONG_REAL_PTR → input number
TEMP_REAL_PTR → output number

Error: D

mqcSHORT_TEMP (Convert SHORT_REAL to TEMP_REAL)

Inputs: SHORT_REAL_PTR → input number
TEMP_REAL_PTR → output number

Error: D

mqcTEMP_LONG (Convert TEMP_REAL to LONG_REAL)

Inputs: TEMP_REAL_PTR → input number
LONG_REAL_PTR → output number

Errors: I,O,U,P

mqcTEMP_SHORT (Convert TEMP_REAL to SHORT_REAL)

Inputs: TEMP_REAL_PTR → input number
SHORT_REAL_PTR → output number

Errors: I,O,U,P

CEL287.LIB

x denotes the 80287 stack top ST.

y denotes the 80287 next stack element ST(1).

STK denotes a 2-byte integer pushed onto the 80286 stack.

All 80286 and 80287 stack inputs are popped on successful return.

The errors columns give the hexadecimal code left in the 11-bit 80287 opcode register, along with the possible errors the function can produce. Unmasked errors trap with the input(s) on the 80287 stack if under the Inputs column; with the output(s) on the 80287 stack if under the Outputs column. The first of the three hex digits tells how many numbers are on the 80287 stack when a trap handler is called.

Name	Function	Errors, trap with: Inputs		Outputs		
mqerACS	x = arc cosine(x)	175	I			
mqerASN	x = arc sine(x)	174	I			
mqerAT2	x = arc tangent(y,x)	277	I	277	U	
mqerATN	x = arc tangent(x)	176	I			
mqerCOS	x = cosine(x)	172	I			
mqerCSH	x = hyperbolic cosine(x)	16F	IO			
mqerDIM	x = max(y−x, +0)	265	I	265	OU	*
mqerEXP	x = e ** x	16B	IOU			
mqerIA2	AX = roundaway(x)	17E	I			
mqerIA4	DXAX = roundaway(x)	168	I			
mqerIAX	x = roundaway(x)	167	I			
mqerIC2	AX = chop(x)	17E	I	17E	P	
mqerIC4	DXAX = chop(x)	179	I	179	P	
mqerICX	x = chop(x)	166	I	166	P	
mqerIE2	AX = roundeven(x)	180	I	180	P	
mqerIE4	DXAX = roundeven(x)	17B	I	17B	P	
mqerIEX	x = roundeven(x)	178	I	178	P	
mqerLGD	x = common log(x)	16D	IZ			
mqerLGE	x = natural log(x)	16C	IZ			
mqerMAX	x = max(x,y)	282	I			*
mqerMIN	x = min(x,y)	281	I			*
mqerMOD	x = (y mod x), has sign(y)	269	I	269	U	
mqerRMD	x = (y mod x), close to 0	27A	I	27A	U	
mqerSGN	x = (y with x's sign)	264	I			*
mqerSIN	x = sine(x)	171	I			
mqerSNH	x = hyperbolic sine(x)	16E	IO			
mqerTAN	x = tangent(x)	173	IZ			
mqerTNH	x = hyperbolic tangent(x)	170	I			
mqerY2X	x = y**x	26A	IZOU			
mqerYI2	x = x**AX	27C	IOU			
mqerYI4	x = x**DXAX	27C	IOU			
mqerYIS	x = x**STK	27C	IOU			

*mqerDIM, mqerMAX, mqerMIN, and mqerSGN can produce D errors from within their interiors.

EH287.LIB

All EH287.LIB procedures operate on a ESTATE287 structure, summarized below.

Elements of ESTATE287:

OPERATION WORD	instruction or procedure that caused error
ARGUMENT BYTE	two nibbles, each coded: 0=no operand 1=ST(0) 2=ST(1) 3=ST(REGISTER) 4=see FORMAT 5=TEMP REAL 6=64-bit integer 7=BCD bit 3 means push once
ARG1(5)WORD	value of bottom-nibble argument
ARG1_FULL BYTE	true if ARG1 contains a value
ARG2(5) WORD	value of top-nibble argument
ARG2_FULL BYTE	true if ARG2 contains a value
RESULT BYTE	formats of result, as in ARGUMENT; except bit 3 on means pop once, bit 7 on means pop twice
RES1(5)WORD	value of bottom-nibble result
RES1_FULL BYTE	true if RES1 contains a value
RES2(5) WORD	value of top-nibble result
RES2_FULL BYTE	true if RES2 contains a value
FORMAT BYTE	format of type 4 ARGUMENT or RESULT: 0=32-bit real 1=32-bit integer 2=64-bit real 3=16-bit integer
REGISTER BYTE	80287 stack register number for type 3 ARGUMENT or RESULT
CONTROL_WORD WORD	80287 control word
STATUS_WORD WORD	80287 status word
TAG_WORD WORD	80287 tag word
ERROR_POINTERS(5) WORD	80287 instruction pointer, opcode, operand pointer
STACK_287(40)WORD	80287 stack of 8 temporary real values

DECODE: Push 4-byte ESTATE287_PTR and 2-byte ERRORS287 before calling. DECODE fills ESTATE287 with information about the 80287 error that caused the exception.

ENCODE: Push 4-byte ESTATE287_PTR, 2-byte ERRORS287, 2-byte CONTROL_WORD, 1-byte RETRY_FLAG before calling. ENCODE restores the 80287 to the state indicated by ESTATE287; if RETRY_FLAG is true if retries the error operation using CONTROL_WORD.

FILTER: Push 2-byte ERRORS287 before calling. FILTER calls DECODE, NORMAL, SIEVE, and ENCODE. FILTER returns AL TRUE if either NORMAL or SIEVE returned TRUE.

NORMAL: Push 4-byte ESTATE287_PTR and 2-byte ERRORS287 before calling. NORMAL returns AL TRUE if D was the only error in ERRORS287; it also normalizes arguments if operation was not a load operation.

SIEVE: Push 4-byte ESTATE287_PTR and 2-byte ERRORS287 before calling. SIEVE returns AL TRUE if a nontrapping NaN occurred that should not have caused an I error.

APPENDIX F
PUBLIC SYMBOLS IN THE SUPPORT LIBRARY

Each of the three libraries DC287.LIB, CEL287.LIB, and EH287.LIB use public symbols other than the names of the procedures documented in this manual. They are internal names, used either in the libraries or by Intel translators.

You should not use any of these names in your programs.

The 80287 emulator or interface libraries have no extra public names other than those listed in Appendix B. The names in Appendix B cannot be generated by Intel translators, eliminating possible conflict.

Following is a list of all public names, both documented and undocumented, for each library.

DC287.LIB

CHK_UNMSKD_O_U_ERR
MQCBINDEC
MQCBIN_DECLOW
MQCDBX
MQCDBXDB
MQCDECBIN
MQCDECBINLO
MQCDECLOW_BIN
MQCDEC_BIN
MQCLONG_TEMP
MQCSHORT_TEMP
MQCSNGXDB
MQCSNX
MQCTEMP_LONG
MQCTEMP_SHORT
MQCXDB
MQCXDBDB
MQCXDBSNG
MQCXSN
POWER_OF_10
UNMSKD_OV_OR_UN
XCPTN_RTRN

CEL287.LIB

MQERACS
MQERAIN
MQERANT
MQERASN
MQERAT2
MQERATN
MQERCI2
MQERCOS
MQERCSH
MQERDIM
MQEREXP
MQERIA2
MQERIA4
MQERIAX
MQERIC2
MQERIC4
MQERICX
MQERIE2
MQERIE4
MQERIEX
MQERINT
MQERIRT
MQERLGD
MQERLGE
MQERMAX
MQERMIN
MQERMOD
MQERNI2
MQERNIN
MQERRI2
MQERRMD
MQERRNT
MQERSGN
MQERSIN
MQERSNH
MQERTAN
MQERTNH
MQERY2X
MQERYI2
MQERYI4
MQERYIS
MQ_1
MQ_2XM1
MQ_63U
MQ_63U1
MQ_63UPI2
MQ_AT2
MQ_CONST
MQ_COS
MQ_CP2N63
MQ_DECIDE
MQ_EXIT
MQ_EXM1
MQ_I
MQ_IRCHK
MQ_LOG
MQ_LOG10
MQ_LOGDN
MQ_MQRPI
MQ_NAN
MQ_NOF
MQ_NORM
MQ_NQ
MQ_OF
MQ_P0
MQ_PI2
MQ_PII
MQ_Q
MQ_RAD
MQ_RERR
MQ_SIN
MQ_TXAM
MQ_U0
MQ_YL2X

EH287.LIB

DECODE
ENCODE
FILTER
FQFORTRANSTATUSCHECK
MQERACS
MQERAIN
MQERANT
MQERASN
MQERAT2
MQERATN
MQERCI2
MQERCOS
MQERCSH
MQERDIM
MQEREXP
MQERINT
MQERIRT
MQERLGD
MQERLGE
MQERMAX
MQERMIN
MQERMOD
MQERN12
MQERNIN
MQERR12
MQERRMD
MQERRNT
MQERSGN
MQERSIN
MQERSNH
MQERTAN
MQERTNH
MQERY2X
MQERY14
NORMAL
SIEVE
TQDECODE87
TQENCODE87
TQFETCH_AND_STORE
TQINSTRUCTION_RETRY
TQNANFILTER
TQNORM87
TQNORMALIZE
TQPOP_THE_TOP
TQREALMATHFILTER
TQRESTORE_PTRS
TQSAVE_PTRS
TQUNPOP_THE_TOP
TQ_312
TQ_320
TQ_322
TQ_324
TQ_326

GLOSSARY OF 80287 AND FLOATING-POINT TERMINOLOGY

This glossary continues the convention of Chapter 6, capitalizing terms that have precise technical meanings. Such terms appear as entries in this glossary. Thus, you may interpret any nonstandard capitalization as a cross-reference.

Affine Mode: a state of the 80287, selected in the 80287 Control Word, in which infinities are treated as having a sign. Thus, the values +INFINITY and −INFINITY are considered different; they can be compared with finite numbers and with each other.

Base: (1) a term used in logarithms and exponentials. In both contexts, it is a number that is being raised to a power. The two equations (y = log base b of x) and (b^y = x) are the same.

Base: (2) a number that defines the representation being used for a string of digits. Base 2 is the binary representation; Base 10 is the decimal representation; Base 16 is the hexadecimal representation. In each case, the Base is the factor of increased significance for each succeeding digit (working up from the bottom).

Bias: the difference between the unsigned Integer that appears in the Exponent field of a Floating-Point Number and the true Exponent that it represents. To obtain the true Exponent, you must subtract the Bias from the given Exponent. For example, the Short Real format has a Bias of 127 whenever the given Exponent is nonzero. If the 8-bit Exponent field contains 10000011, which is 131, the true Exponent is 131−127, or +4.

Biased Exponent: the Exponent as it appears in a Floating-Point Number, interpreted as an unsigned, positive number. In the above example, 131 is the Biased Exponent.

Binary Coded Decimal: a method of storing numbers; it retains a base 10 representation. Each decimal digit occupies four full bits (one hexadecimal digit). The hex values A through F (1010 through 1111) are not used. The 80287 supports a "Packed Decimal" format that consists of nine bytes of Binary Coded Decimal (eighteen decimal digits), and one sign byte.

Binary Point: an entity just like a decimal point except that it exists in binary numbers. Each binary digit to the right of the Binary Point is multiplied by an increasing negative power of two.

C3—C0: the four "condition code" bits of the 80287 Status Word. These bits are set to certain values by the compare, test, examine, and remainder functions of the 80287.

Characteristic: a term used for some non-Intel computers. It denotes the Exponent field of a Floating-Point Number.

Chop: to set the fractional part of a real number to zero, yielding the nearest integer in the direction of zero.

Control Word: a 16-bit 80287 register the user can set to determine the modes of computation the 80287 will use and the error interrupts that will be enabled.

Denormal: a special form of Floating-Point Number. It is produced when an Underflow occurs. On the 80287, a Denormal is defined as a number with a Biased Exponent that is zero. By providing a Significand with leading zeroes, the range of possible negative Exponents can be extended by the number of bits in the Significand. Each leading zero is a bit of lost accuracy, so the extended Exponent range is obtained by reducing significance.

Double Extended: the Standard's term for the 80287 Temporary Real format, with more Exponent and Significand bits than the Double (Long Real) format and an explicit Integer bit in the Significand.

Double Floating-Point Number: the Standard's term for the 80287's 64-bit Long Real format.

Environment: the 14 bytes of 80287 registers affected by the FSTENV and FLDENV instructions. It encompasses the entire state of the 80287, except for the 8 Temporary Real numbers of the 80287 stack. Included are the Control Word, Status Word, Tag Word, and the instruction, opcode, and operand information provided by interrupts.

Exception: any of the six error conditions (I, D, O, U, Z, P) signalled by the 80287.

Exponent: (1) any power that is raised by an exponential function. For example, the operand to the function mqerEXP is an Exponent. The Integer operand to mqerYI2 is an Exponent. (2) the field of a Floating-Point Number. It indicates the magnitude of the number. This would fall under the above more general definition (1) except that a Bias sometimes needs to be subtracted to obtain the correct power.

Floating-Point Number: a sequence of data bytes that, when interpreted in a standardized way, represents a Real number. Floating-Point Numbers are more versatile than Integer representations because they include fractions, and their Exponent parts allow a much wider range of magnitude than possible with fixed-length Integer representations.

Gradual Underflow: a method of handling the Underflow error condition. It minimizes the loss of accuracy in the result. If a Denormal number represents the correct result, that Denormal is returned. Thus, digits are lost only to the extent of denormalization. Most computers return zero when Underflow occurs, losing all significant digits.

Implicit Integer Bit: a part of the Significand in the Short Real and Long Real formats. It is not explicitly given. In these formats, the entire given Significand is considered to be to the right of the Binary Point. A single Implicit Integer Bit to the left of the Binary Point is always 1, except in one case: when the Exponent is the minimum (Biased Exponent is 0), the Implicit Integer Bit is 0.

Indefinite: a special value returned by functions when no other sensible answer is possible. Each Floating-Point format has one Nontrapping NaN that is designated as the Indefinite value. For binary Integer formats, the negative number farthest from zero is often considered the Indefinite value. For the 80287 Packed Decimal format, the Indefinite value contains all 1's in the sign byte and the uppermost digits byte.

Infinity: a value that has greater magnitude than any Integer or Real number. The existence of Infinity is subject to heated philosophical debate. However, it is often useful to consider Infinity as another number, subject to special rules of arithmetic. All three Intel Floating-Point formats provide representations for +INFINITY and −INFINITY. They support two ways of dealing with Infinity: Projective (unsigned) and Affine (signed).

Instruction Offset: See Instruction Pointer.

Instruction Pointer: in real mode this is the 20-bit physical address and 11-bit opcode. In protected mode this is the 32-bit virtual address used by the program that executed an ESC instruction.

Integer: a finite number (positive, negative, or zero) that has no fractional part. "Integer" can also mean the computer representation for such a number: a sequence of data bytes interpreted in a standard way. It is perfectly reasonable for Integers to be represented in a Floating-Point format; this is what the 80287 does whenever an Integer is pushed onto the 80287 stack.

Invalid Operation: the error condition for the 80287. It covers all cases not covered by other errors. Included are 80287 stack overflow and underflow, NaN inputs, illegal infinite inputs, out-of-range inputs, and illegal unnormal inputs.

Long Integer: an Integer format supported by the 80287. It consists of a 64-bit Two's Complement quantity.

Long Real: a Floating-Point format supported by the 80287. It consists of a sign, an 11-bit Biased Exponent, an Implicit Integer Bit, and a 52-bit Significand–a total of 64 explicit bits.

Mantissa: a term used for some non-Intel computers. It denotes the Significand of a Floating-Point Number.

Masked: a term that applies to each of the six 80287 Exceptions (I,D,Z,O,U,P). An exception is Masked if a corresponding bit in the 80287 Control Word is set to 1. If an exception is Masked, the 80287 will not generate an interrupt when the error condition occurs; instead, it will provide its own error recovery.

NaN: an abbreviation for "Not a Number"; a Floating-Point quantity that represents no numeric or infinite quantity. NaNs should be returned by functions that encounter serious errors. If created during a sequence of calculations, they are transmitted to the final answer; they can contain information about where the error occurred.

NDP: Numeric Data Processor. This is any iAPX 286 system that contains an 80287 or the full 80287 emulator.

Nontrapping NaN: a NaN in which the most significant bit of the fractional part of the Significand is 1. By convention, these NaNs can undergo certain operations without visible error. Nontrapping NaNs are implemented for the 80287 via the software in EH287.LIB.

Normal: the representation of a number in a Floating-Point format in which the Significand has an Integer bit 1 (either explicit or Implicit).

Normalizing Mode: a state in which nonnormal inputs are automatically converted to normal inputs whenever they are used in arithmetic. Normalizing Mode is implemented for the 80287 via the software in EH287.LIB.

NPX: Numeric Processor Extension. This is the 80287.

Overflow: an error condition in which the correct answer is finite, but has magnitude too great to be represented in the destination format.

Packed Decimal: an Integer format supported by the 80287. A Packed Decimal number is a 10-byte quantity, with nine bytes of 18 Binary Coded Decimal digits and one byte for the sign.

Pop: to remove from a stack the last item placed on the stack.

Precision Control: an option programmed through the 80287 Control Word. It allows all 80287 arithmetic to be performed with reduced precision. Since no speed advantage results from this option, its only use is for strict compatibility with the Standard and with other computer systems.

Precision Exception: an 80287 error condition that results when a calculation does not return an exact answer. This exception is usually Masked and ignored; it is used only in extremely critical applications when the user must know if the results are exact.

Projective Mode: a state of the 80287, selected in the 80287 Control Word, in which infinities are treated as not having a sign. Thus, the values +INFINITY and −INFINITY are considered the same. Certain operations such as comparison to finite numbers are illegal in Projective Mode, but legal in Affine Mode. Thus, Projective Mode gives you a greater degree of error control over infinite inputs.

Pseudo-Zero: a special value of the Temporary Real format. It is a number with a zero significand and an Exponent that is neither all zeroes or all ones. Pseudo-Zeroes can result from multiplying two Unnormal numbers; they are very rare.

Real: any finite value (negative, positive, or zero) that can be represented by a decimal expansion. The fractional part of the decimal expansion can contain an infinite number of digits. Reals can be represented as the points of a line marked off like a ruler. The term "Real" can also refer to a Floating-Point Number that represents a Real value.

Short Integer: an Integer format supported by the 80287. It consists of a 32-bit Two's Complement quantity. Short Integer is not the shortest 80287 Integer format—the 16-bit Word Integer is.

Short Real: a Floating-Point Format supported by the 80287. It consists of a sign, an 8-bit Biased Exponent, an Implicit Integer Bit, and a 23-bit Significand; a total of 32 explicit bits.

Significand: the part of a Floating-Point Number that consists of the most significant nonzero bits of the number, if the number were written out in an unlimited binary format. The Significand alone is considered to have a Binary Point after the first (possibly Implicit) bit; the Binary Point is then moved according to the value of the Exponent.

Single Extended: a Floating-Point format required by the Standard. It provides greater precision than Single; it also provides an explicit Integer Significand bit. The 80287's Temporary Real format meets the Single Extended requirement as well as the Double Extended requirement.

Single Floating-Point Number: the Standard's term for the 80287's 32-bit Short Real format.

Standard: "a Proposed Standard for Binary Floating-Point Arithmetic," Draft 10.0 of IEEE Task P754, December 2, 1982

Status Word: A 16-bit 80287 register that can be manually set, but that is usually controlled by side effects to 80287 instructions. It contains condition codes, the 80287 stack pointer, busy and interrupt bits, and error flags.

Tag Word: a 16-bit 80287 register that is automatically maintained by the 80287. For each space in the 80287 stack, it tells if the space is occupied by a number; if so, it gives information about what kind of number.

Temporary Real: the main Floating-Point format used by the 80287. It consists of a sign, a 15-bit Biased Exponent, and a Significand with an explicit Integer bit and 63 fractional-part bits.

Transcendental: one of a class of functions for which polynomial formulas are always approximate, never exact for more than isolated values. The 80287 supports trigonometric, exponential, and logarithmic functions; all are Transcendental.

Trapping NaN: a NaN that causes an I error whenever it enters into a calculation or comparison, even a nonordered comparison.

Two's Complement: a method of representing Integers. If the uppermost bit is 0, the number is considered positive, with the value given by the rest of the bits. If the uppermost bit is 1, the number is negative, with the value obtained by subtracting ($2^{\text{bit count}}$) from all the given bits. For example, the 8-bit number 11111100 is −4, obtained by subtracting 2^8 from 252.

Unbiased Exponent: the true value that tells how far and in which direction to move the Binary Point of the Significand of a Floating-Point Number. For example, if a Short Real Exponent is 131, subtract the Bias 127 to obtain the Unbiased Exponent +4. Thus, the Real number being represented is the Significand with the Binary Point shifted four bits to the right.

Underflow: an error condition in which the correct answer is nonzero, but has a magnitude too small to be represented as a Normal number in the destination Floating-Point format. The Standard specifies that an attempt be made to represent the number as a Denormal.

Unmasked: a term that applies to each of the six 80287 Exceptions (I,D,Z,O,U,P). An exception is Unmasked if a corresponding bit in the 80287 Control Word is set to 0. If an exception is Unmasked, the 80287 generates an interrupt when the error condition occurs. You can provide an interrupt routine that customizes your error recovery.

Unnormal: a Temporary Real representation in which the explicit Integer bit of the Significand is zero and the exponent is nonzero. Unnormal numbers as are considered distinct from Denormal numbers.

Word Integer: an Integer format supported by both the 80286 and the 80287. It consists of a 16-bit Two's Complement quantity.

Zerodivide: an error condition in which the inputs are finite, but the correct answer, even with an unlimited exponent, has infinite magnitude.

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating).

NAME ______ DATE ______

TITLE ______

COMPANY NAME/DEPARTMENT ______

ADDRESS ______

CITY ______ STATE ______ ZIP CODE ______

(COUNTRY)

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS ...

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

NO POSTAGE
NECESSARY
IF MAILED
IN U.S.A.

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1040 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
Attn: Technical Publications M/S 6-2000
3065 Bowers Avenue
Santa Clara, CA 95051

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 987-8080

Printed in U.S.A.